

# MATEMÁTICAS PARA PROFUNDO APRENDIENDO

PRACTICANTE  
DOMINAR LAS REDES NEURALES

SGUIDETO

RONALD T. KNEUSEL









# MATEMÁTICAS PARA EL APRENDIENDO

Lo que necesitas saber para  
Comprender las redes neuronales

por Ronald T. Kneusel



San Francisco

MATEMÁTICAS PARA EL APRENDIZAJE PROFUNDO. Copyright © 2022 por Ronald T. Kneusel.

Reservados todos los derechos. Ninguna parte de este trabajo puede reproducirse o transmitirse de ninguna forma ni por ningún medio, electrónico o mecánico, incluidas fotocopias, grabaciones o cualquier sistema de almacenamiento o recuperación de información, sin el permiso previo por escrito del propietario de los derechos de autor y del editor.

ISBN-13: 978-1-7185-0190-4 (impreso)

ISBN-13: 978-1-7185-0191-1 (libro electrónico)

Editor: William Pollock Gerente de

producción: Rachel Monaghan Editores de producción:

Dapinder Dosanjh y Katrina Taylor Editor de desarrollo: Alex Freed Ilustrador

de portada: James L. Barry Diseño de portada

y interiores: Octopod Studios Revisor

técnico: David Gorodetzky Corrector de estilo: Carl Quesnel

Corrector de pruebas: Emelie Battaglia

Para obtener información sobre distribuidores de libros o traducciones, comuníquese directamente con No Starch Press, Inc.: No Starch Press, Inc. 245 8th

Street, San Francisco, CA 94103 teléfono:

415.863.9900; fax: 415.863.9950; info@nostarch.com; [www.nostarch.com](http://www.nostarch.com)

Número de control de la Biblioteca del Congreso: 2021939724

No Starch Press y el logotipo de No Starch Press son marcas comerciales registradas de No Starch Press, Inc. Otros nombres de productos y empresas mencionados aquí pueden ser marcas comerciales de sus respectivos propietarios. En lugar de utilizar un símbolo de marca registrada cada vez que aparece un nombre de marca registrada, utilizamos los nombres solo de manera editorial y para beneficio del propietario de la marca, sin intención de infringir la marca.

La información contenida en este libro se distribuye "tal cual", sin garantía. Si bien se han tomado todas las precauciones en la preparación de este trabajo, ni el autor ni No Starch Press, Inc. tendrán responsabilidad alguna ante ninguna persona o entidad con respecto a cualquier pérdida o daño causado o presuntamente causado directa o indirectamente por el información contenida en el mismo.

En memoria de Tom “Fitz” Fitzpatrick (1944–2013), el mejor profesor de matemáticas que he tenido. Y a todos los profesores de matemáticas: reciben muy poco reconocimiento por todo su arduo trabajo.



## Sobre el Autor

Ron Kneusel ha estado trabajando con aprendizaje automático en la industria desde 2003 y completó un doctorado en aprendizaje automático de la Universidad de Colorado, Boulder, en 2016. Ron tiene otros tres libros: Aprendizaje profundo práctico: una introducción basada en Python (Sin prensa de almidón), Números y computadoras (Springer) y Números aleatorios y computadoras (Springer).

## Acerca del revisor técnico

David Gorodetzky es un científico investigador que trabaja en la intersección de la teledetección y el aprendizaje automático. Desde 2011 ha dirigido un pequeño grupo de investigación dentro de una gran empresa de ingeniería de servicios gubernamentales que desarrolla soluciones de aprendizaje profundo para una amplia variedad de problemas de teledetección. David comenzó su carrera en geología planetaria y geofísica, se desvió hacia la consultoría ambiental, luego estudió la reconstrucción del paleoclima a partir de núcleos de hielo polar en la escuela de posgrado, antes de dedicarse a una carrera en teledetección por satélite. Durante más de 15 años fue consultor principal de un grupo de servicios de software que desarrollaba análisis de imágenes y algoritmos de procesamiento de señales para clientes de diversos campos, incluidos el aeroespacial, la agricultura de precisión, el reconocimiento, la biotecnología y los cosméticos.



## CONTENIDOS BREVES

Prólogo .....	xvii
Agradecimientos .....	xxi
Introducción .....	xxiii
Capítulo 1: Preparando el escenario .....	1
Capítulo 2: Probabilidad .....	17
Capítulo 3: Más probabilidad .....	41
Capítulo 4: Estadísticas .....	67
Capítulo 5: Álgebra lineal .....	103
Capítulo 6: Más álgebra lineal .....	127
Capítulo 7: Cálculo diferencial .....	163
Capítulo 8: Cálculo matricial .....	193
Capítulo 9: Flujo de datos en redes neuronales .....	221
Capítulo 10: Propagación hacia atrás .....	243
Capítulo 11: Descenso de gradiente .....	271
Apéndice: Yendo más allá .....	305
Índice .....	309



# CONTENIDODEDETALLE

PREFACIO	xvii
EXPRESIONES DE GRATITUD	xxi
INTRODUCCIÓN	xxiii
¿Para quién es este libro? .....	XXIV
Sobre este libro .....	XXIV
 1	
PREPARANDO EL ESCENARIO	1
Instalación de los kits de herramientas .....	2
Linux.....	2
Mac OS .....	3
Ventanas .....	3
NumPy .....	4
Definición de matrices .....	5
Tipos de datos .....	5
Matrices 2D .....	6
Ceros y Unos .....	7
Indexación avanzada .....	7
Lectura y escritura en disco .....	10
Ciencia ficción .....	11
Matplotlib .....	12
Scikit-Aprende .....	14
Resumen .....	15
 2	
PROBABILIDAD	17
Conceptos básicos .....	18
Espacio muestral y eventos.....	18
Variables aleatorias .....	19
"Los humanos son malos en cuanto a probabilidades" .....	19
Las reglas de la probabilidad .....	21
Probabilidad de un evento .....	21
Regla de la suma .....	24
Regla del producto .....	25

"Regla de la suma revisada " .....	25 La
paradoja del cumpleaños .....	26
Probabilidad condicional .....	30 Probabilidad
total .....	31 Probabilidad conjunta y
marginal .....	32 Tablas de probabilidad
conjunta .....	33 Regla de la cadena para la
probabilidad .....	37
Resumen .....	39
 3	
<b>MÁS PROBABILIDAD</b>	
Distribuciones de probabilidad .....	41
Histogramas y probabilidades .....	42 Distribuciones
de probabilidad discreta .....	45 Distribuciones de
probabilidad continua .....	51 Teorema del límite
central .....	55 La ley de los grandes
números .....	58 Teorema de
Bayes .....	59 Cáncer o no
Redux .....	60 Actualización del
Prior .....	61 Teorema de Bayes en
aprendizaje automático .....	62
Resumen .....	sesenta y cinco
 4	
<b>ESTADÍSTICAS</b>	
Tipos de datos .....	68 Datos
Nominales .....	68 Datos
ordinales .....	68 datos de
intervalo .....	68 Datos de
proporción .....	68 Uso de
datos nominales en aprendizaje profundo .....	69 Estadísticas
resumidas .....	70 Medias y
mediana .....	70 medidas de
variación .....	74 cuantiles y diagramas de
caja .....	78 datos
faltantes .....	83
Correlación .....	85 Correlación
de Pearson .....	86 Correlación de
Spearman .....	90 Pruebas de
hipótesis .....	92
hipótesis .....	93 La prueba
t .....	95 La prueba U de
Mann-Whitney .....	99
Resumen .....	102

<b>5</b>		
<b>ÁLGEBRA LINEAL</b>		<b>103</b>
Escalares, Vectores, Matrices y Tensores . . . . .	104	
escalares . . . . .	104	
vectores . . . . .	104	
matrices . . . . .	105	
Tensores . . . . .	106	
Aritmética con Tensores . . . . .	109	
Operaciones con matrices . . . . .		
109 Operaciones vectoriales . . . . .	111	
Multiplicación de matrices . . . . .		
120 Producto Kronecker . . . . .		
125 Resumen . . . . .	126	
<b>6</b>		
<b>MÁS ÁLGEBRA LINEAL</b>		<b>127</b>
Matrices cuadradas . . . . .	128	
¿Por qué matrices cuadradas? . . . . .		
128 Transposición, traza y potencias . . . . .		
129 Matrices Cuadradas Especiales . . . . .		
131 La matriz de identidad . . . . .		
132 Determinantes . . . . .		
134 Inversas . . . . .		
137 Matrices simétricas, ortogonales y unitarias . . . . .	139	
Definitividad de una matriz simétrica . . . . .	140	
Vectores propios y valores propios . . . . .	141	
Encontrar valores propios y vectores propios . . . . .	141	
Normas vectoriales y métricas de distancia . . . . .	144	
Normas L y métricas de distancia . . . . .	145	
Matrices de covarianza . . . . .	146	
Distancia de Mahalanobis . . . . .	148	
Divergencia Kullback-Leibler . . . . .	151	
Análisis de componentes principales . . . . .		
153 Descomposición de valores singulares y pseudoinversa . . . . .		
157 SVD en acción . . . . .		
158 Dos aplicaciones . . . . .	159	
Resumen . . . . .	161	
<b>7</b>		
<b>CALCULO DIFERENCIAL</b>		<b>163</b>
Pendiente . . . . .		
164 Derivados . . . . .		
165 Una definición formal . . . . .		
165 reglas básicas . . . . .		
167 reglas para funciones trigonométricas . . . . .	172	

Reglas para exponentiales y logaritmos .....	175
Mínimos y máximos de funciones .....	177
Derivados Parciales .....	181
Derivados Parciales Mixtos .....	183 La
regla de la cadena para derivados parciales .....	184
degradados .....	186
Calcular el gradiente .....	186
Visualizando el gradiente .....	189
Resumen .....	191
 8	
<b>CÁLCULO MATRIZ</b>	193
Las Fórmulas .....	194 Una
función vectorial mediante un argumento escalar .....	194
Una función escalar mediante un argumento vectorial .....	
196 Una función vectorial mediante un vector .....	
197 Una función matricial mediante un escalar .....	
198 Una función escalar mediante una matriz .....	
198 Las identidades .....	
199 Una función escalar mediante un vector .....	
199 Una función vectorial mediante un escalar .....	
202 Una función vectorial mediante un vector .....	
203 Una función escalar mediante una matriz .....	
203 jacobianos y hessianos .....	205
Sobre los jacobianos .....	205 Sobre
los hessianos .....	211 Algunos
ejemplos de derivadas de cálculo matricial .....	217 Derivada de
operaciones por elementos .....	217 Derivada de la
Función de Activación .....	218
Resumen .....	220
 9	
<b>FLUJO DE DATOS EN REDES NEURALES</b>	221
Representando datos .....	222
Redes neuronales tradicionales .....	222
Redes convolucionales profundas .....	223
Flujo de datos en redes neuronales tradicionales .....	225
Flujo de datos en redes neuronales convolucionales .....	229
Convolución .....	229
capas convolucionales .....	234
capas de agrupación .....	
237 capas completamente conectadas .....	
239 Flujo de datos a través de una red neuronal convolucional .....	239
Resumen .....	242

10		
<b>RETROPAGACIÓN</b>		<b>243</b>
¿Qué es la retropropagación? . . . . .	244	
Propagación hacia atrás a mano . . . . .	245	
Cálculo de las derivadas parciales . . . . .	246	
Traduciendo a Python . . . . .	249	
Entrenamiento y prueba del modelo . . . . .	253	
Propagación hacia atrás para redes totalmente conectadas . . . . .	254	
Propagación hacia atrás del error . . . . .	255	
Cálculo de las derivadas parciales de las ponderaciones y sesgos . . . . .	258	Una implementación de Python . . . . .
de la implementación . . . . .	260	Uso
computacionales . . . . .	264	Gráficos
Resumen . . . . .	267	
	269	
11		
<b>DESCENSO DE GRADIENTE</b>		<b>271</b>
La idea básica . . . . .	272	
Descenso de gradiente en una dimensión . . . . .	272	
Descenso de gradiente en dos dimensiones . . . . .	276	
Descenso de gradiente estocástico . . . . .	282	
Impulso . . . . .	284	¿Qué es el impulso? . . . . .
1D . . . . .	285	Momento en
2D . . . . .	287	Impulso en
entrenamiento con Momentum . . . . .	289	Modelos de
Nesterov . . . . .	294	Impulso de
gradiente adaptativo . . . . .	297	Descenso de
RMSprop . . . . .	297	
Adagrad y Adadelta . . . . .	299	
Adán . . . . .	300	
Algunas reflexiones sobre los optimizadores . . . . .		
301 Resumen . . . . .	303	
Epílogo . . . . .	303	
<b>APÉNDICE: IR MÁS ALLÁ</b>		<b>305</b>
Probabilidades y estadísticas . . . . .	305	
Álgebra lineal . . . . .	306	
Cálculo . . . . .	306	
Aprendizaje profundo . . . . .	307	
<b>ÍNDICE</b>		<b>309</b>



## PARA ORDEN EW

La inteligencia artificial (IA) está en todas partes. No necesitas buscar más allá del dispositivo en su bolsillo como evidencia: su teléfono ahora ofrece seguridad de reconocimiento facial, obedece comandos de voz simples y desenfoca fondos digitalmente en tus selfies y aprende silenciosamente tus intereses para brindarte una visión personalizada. Los modelos de IA se están utilizando para analizar montañas de datos para crear vacunas de manera eficiente, mejorar la manipulación robótica y construir sistemas autónomos. vehículos, aprovechar el poder de la computación cuántica e incluso adaptarse a sus dominio del ajedrez en línea. La industria se está adaptando para garantizar la última tecnología. Las capacidades de IA se pueden integrar en su dominio de experiencia, y el mundo académico es construir un plan de estudios que exponga conceptos de inteligencia artificial a cada uno disciplina basada en títulos. Se acerca una era de autonomía cognitiva impulsada por máquinas. sobre nosotros, y si bien todos somos consumidores de IA, aquellos que expresan interés En su desarrollo es necesario comprender a qué se debe su sustancial crecimiento durante la última década. Aprendizaje profundo, una subcategoría de máquina aprendizaje, aprovecha redes neuronales muy profundas para modelar sistemas complicados que históricamente han planteado problemas a los métodos analíticos tradicionales. A El nuevo uso práctico de estas redes neuronales profundas es directamente responsable de este aumento en el desarrollo de la IA, un concepto que la mayoría atribuiría a Alan Turing allá por los años cincuenta. Pero si el aprendizaje profundo es el motor de la IA, ¿Cuál es el motor del aprendizaje profundo?

El aprendizaje profundo se basa en muchos conceptos importantes de los campos de la ciencia, la tecnología, la ingeniería y las matemáticas (STEM). Los reclutadores de la industria continúan buscar una definición formal de sus componentes mientras intentan atraer a los mejores talentos con solicitudes de trabajo más descriptivas. De manera similar, los coordinadores de programas académicos tienen la tarea de desarrollar el plan de estudios que desarrolle esta habilidad. puesto que permea todas las disciplinas. Si bien es inherentemente interdisciplinario en

En la práctica, el aprendizaje profundo se basa en principios matemáticos básicos de probabilidad y estadística, álgebra lineal y cálculo. El grado

Hasta qué punto un individuo debe abrazar y comprender estos principios depende del nivel de intimidad que se espera tener con las tecnologías de aprendizaje profundo.

Para el implementador, Math for Deep Learning actúa como una herramienta de solución de problemas. Guía para los desafíos inevitables que se encuentran en la implementación de redes neuronales profundas. Este individuo generalmente se preocupa por la implementación eficiente de soluciones preexistentes con tareas que incluyen la identificación y adquisición de código fuente abierto, creación de un entorno de trabajo adecuado, ejecutar cualquier prueba unitaria disponible y, finalmente, volver a capacitarse con datos relevantes para la aplicación de intereses. Estas redes neuronales profundas pueden contener decenas o cientos de millones de parámetros que se pueden aprender, y suponiendo una adecuada competencia del usuario, la optimización exitosa depende de hiperparámetros sensibles selección y acceso a datos de formación que representen suficientemente a la población. El primer (y segundo, y tercer) intento de implementación a menudo requiere un viaje desalentador hacia la interrogación de redes neuronales, que requiere disección y comprensión de alto nivel de los impulsos matemáticos presentado aquí.

En algún momento, el implementador suele convertirse en el integrador. Este nivel de experiencia requiere cierta familiaridad con el dominio de aplicación deseado y una comprensión de nivel inferior de los componentes básicos que permiten aprendizaje profundo. Además de los desafíos enfrentados en la implementación básica, el integrador debe poder generalizar conceptos básicos para moldear un modelo matemático al dominio deseado. ¡El desastre vuelve a ocurrir! Quizás el individuo experimente el problema del gradiente explosivo. Quizás el integrador desea una función de pérdida más representativa que pueda plantear problemas de diferenciabilidad. O tal vez, durante el entrenamiento, el individuo reconoce que el seleccionado La estrategia de optimización es ineficaz para el problema. Matemáticas para el aprendizaje profundo llena un vacío dentro de la comunidad al ofrecer una visión general coherente de los conceptos matemáticos críticos que componen el aprendizaje profundo y ayuda a superar estos obstáculos.

El integrador se convierte en innovador cuando se siente cómodo con el tema. La materia permite al individuo ser verdaderamente creativo. Con la innovación viene el Necesidad de difusión de información, que a menudo requiere tiempo fuera del desarrollo práctico para publicación, presentación y una buena cantidad de enseñanza. Math for Deep Learning sirve como manual para la base de que el Innovador tiene en alta estima y proporciona referencias rápidas y recordatorios. de semillas que produzcan nuevos avances en inteligencia artificial.

Así como estos roles se complementan entre sí, el aprendizaje profundo crea sus propios Jerarquía, una de conceptos o características no intuitivas que resuelven una tarea específica. El alcance del problema puede ser abrumador sin una atención dedicada enfocar. El Dr. Kneusel tiene más de 15 años de experiencia en la industria aplicando el aprendizaje automático y el aprendizaje profundo a la generación y explotación de imágenes. problemas, y creó Math for Deep Learning para consolidar y enfatizar lo que más importa: la base matemática a partir de la cual todas las soluciones de redes neuronales son posibles. Ningún libro de texto está completo y este

uno presenta otros recursos que exponen temas de estadística, álgebra lineal y cálculo. *Math for Deep Learning* es para personas que buscan una descripción general autónoma y concentrada de los componentes que construyen el motor matemático de la herramienta principal de la IA.

Dr. Derek J. Walvoord



## RECONOCER LEDGMENTOS

Soy un Oso de Muy Poco Cerebro, y las palabras largas me molestan.  
-Winnie the Pooh

Este libro no es sólo el resultado de mis propios esfuerzos. Un sincero agradecimiento y reconocimiento están en orden.

Primero, quiero agradecer a toda la excelente gente de No Starch Press por la oportunidad de trabajar con ellos nuevamente. Todos ellos son profesionales genuinamente consumados y es un placer interactuar con ellos, y eso se duplica para mi editor, Alex Freed. Una vez más, ella tomó mi prosa divagante y la perfeccionó hasta convertirla en algo claro y coherente.

También quiero agradecer a mi amigo David Gorodetzky por su experta revisión técnica. Las sugerencias de David y su forma sutil de señalar los errores han fortalecido el libro. Si persisten errores, es enteramente culpa mía por no ser lo suficientemente sabio como para escuchar el sabio consejo de David.



## INTRODUCCIÓN



Las matemáticas son esenciales para el mundo moderno.

El aprendizaje profundo también se está volviendo rápidamente básico. De la promesa de la conducción autónoma

desde automóviles hasta sistemas médicos que detectan fracturas mejor que todos los médicos, excepto los mejores, por no decir nada.

Con asistentes controlados por voz cada vez más capaces, y posiblemente preocupantes, el aprendizaje profundo está en todas partes.

Este libro cubre las matemáticas esenciales para hacer comprensible el aprendizaje profundo. Es cierto que puedes aprender los kits de herramientas, configurar los archivos de configuración o el código Python, formatear algunos datos y entrenar un modelo, todo sin entender lo que estás haciendo, y mucho menos las matemáticas detrás de esto. Y, gracias al poder del aprendizaje profundo, a menudo tendrá éxito. Sin embargo, No lo entenderás y no deberías estar satisfecho. Para entender, tu Necesito algo de matemáticas. No muchas matemáticas, pero sí algunas matemáticas específicas. En particular, Necesitará conocimientos prácticos sobre temas de probabilidad, estadística, álgebra lineal y cálculo diferencial. Afortunadamente, esos son los temas que aborda este libro. sucede con la dirección.

## ¿Para quién es este libro?

Este no es un libro introductorio al aprendizaje profundo. No te enseñará el Conceptos básicos del aprendizaje profundo. Más bien, pretende ser un complemento de dicho libro. (Consulte mi libro Aprendizaje profundo práctico: una introducción basada en Python [sin almidón Press, 2021]. Espero que esté familiarizado con el aprendizaje profundo, al menos conceptualmente, aunque le explicaré las cosas a lo largo del camino.

Además, espero que aporte ciertos conocimientos. I

Espero que sepas matemáticas de la escuela secundaria, en particular álgebra. Yo también Esperamos que estés familiarizado con la programación usando Python, R o similar. idioma. Usaremos Python 3.x y algunos de sus kits de herramientas populares, como como NumPy, SciPy y scikit-learn.

He intentado mantener otras expectativas al mínimo. Después de todo, el El objetivo del libro es brindarle lo que necesita para tener éxito en profundidad. aprendiendo.

## Sobre este libro

En esencia, este es un libro de matemáticas. Pero en lugar de pruebas y ejercicios de práctica, Usaremos código para ilustrar los conceptos. El aprendizaje profundo es una disciplina aplicada que es necesario realizar para poder comprender. Por lo tanto, usaremos código para cerrar la brecha entre el conocimiento matemático puro y la práctica.

Los capítulos se construyen uno sobre el otro, con capítulos fundamentales seguidos. respaldado por temas matemáticos más avanzados y, en última instancia, algoritmos de aprendizaje profundo que hacen uso de todo lo tratado en los capítulos anteriores. Recomiendo leer el libro de principio a fin y, si lo deseas, saltarte temas. ya estás familiarizado cuando los encuentras.

Capítulo 1: Preparando el escenario Este capítulo configura nuestro entorno de trabajo y los kits de herramientas que usaremos, que son los que se usan con más frecuencia en aprendizaje profundo.

Capítulo 2: Probabilidad La probabilidad afecta a casi todos los aspectos de la investigación profunda. aprendizaje y es esencial para comprender cómo aprenden las redes neuronales. Este capítulo, el primero de dos sobre este tema, presenta aspectos fundamentales Temas en probabilidad.

Capítulo 3: Más probabilidad La probabilidad es tan importante que uno El capítulo no es suficiente. Este capítulo continúa nuestra exploración e incluye temas clave de aprendizaje profundo, como distribuciones de probabilidad y métodos de Bayes. teorema.

Capítulo 4: Estadísticas Las estadísticas dan sentido a los datos y son cruciales para evaluar modelos. Las estadísticas van de la mano con la probabilidad, por lo que Es necesario comprender las estadísticas para comprender el aprendizaje profundo.

Capítulo 5: Álgebra lineal El álgebra lineal es el mundo de los vectores y matrices. El aprendizaje profundo está, en esencia, centrado en el álgebra lineal. La implementación de redes neuronales es un ejercicio de matemática vectorial y matricial, por lo que es esencial comprender lo que estos conceptos representan y cómo trabajar con ellos.

Capítulo 6: Más álgebra lineal Este capítulo continúa nuestra exploración del álgebra lineal, enfocándose en temas importantes relacionados con matrices.

Capítulo 7: Cálculo diferencial Quizás el concepto más fundamental detrás del entrenamiento de redes neuronales sea el gradiente. Para entender el gradiente, qué es y cómo utilizarlo, debemos saber cómo trabajar con derivadas de funciones. Este capítulo sienta las bases necesario comprender las derivadas y los gradientes.

Capítulo 8: Cálculo matricial El aprendizaje profundo manipula derivados de vectores y matrices. Por lo tanto, en este capítulo generalizamos el concepto de derivada a estos objetos.

Capítulo 9: Flujo de datos en redes neuronales Para comprender cómo funcionan las redes neuronales. Las redes manipulan vectores y matrices, necesitamos entender cómo los datos fluyen a través de la red. Ése es el tema de este capítulo.

Capítulo 10: Retropropagación Entrenamiento exitoso de redes neuronales Generalmente involucra dos algoritmos que van de la mano: retropropagación. y descenso de gradiente. En este capítulo, analizamos la propagación hacia atrás en detalle para ver cómo se aplican las matemáticas que aprendimos anteriormente en el libro. al entrenamiento de redes neuronales reales.

Capítulo 11: Descenso de gradiente El descenso de gradiente utiliza los gradientes que proporciona el algoritmo de retropropagación para entrenar una red neuronal. Este capítulo explora el descenso de gradientes, comenzando con ejemplos 1D. y progresar hacia redes neuronales completamente conectadas. También describe y compara variantes comunes del descenso de gradientes.

Apéndice: ir más allá Es necesario pasar por alto muchos temas de probabilidad, estadística, álgebra lineal y cálculo. este apéndice le indica recursos que le ayudarán a ir más allá con el Las matemáticas detrás del aprendizaje profundo.

Puedes descargar todo el código del libro aquí: <https://github.com/rkneusel9/MathForDeepLearning/>. Y por favor mire <https://nostarch.com/aprendizaje profundo de matemáticas / para futuras erratas. Empecemos.>



# 1

## PREPARANDO EL ESCENARIO



Aunque este libro no tiene un enfoque tradicional ejercicios de matemáticas, necesitamos jugar con los conceptos si queremos dominarlos.

Tendremos muchas oportunidades para eso, pero En lugar de ejercicios con lápiz y papel, usaremos código.

Este capítulo le ayudará a preparar el escenario configurando nuestro entorno de trabajo. A lo largo del libro, trabajaré en Linux, específicamente Ubuntu.

20.04, aunque lo que estamos haciendo probablemente funcionará en versiones posteriores de Ubuntu y la mayoría de las otras distribuciones de Linux también. Para completar, he Se incluyen secciones sobre la configuración de entornos macOS y Windows. I Cabe señalar que el sistema operativo esperado para el aprendizaje profundo es Linux, y la mayoría de las cosas también funcionan en macOS. Windows suele ser una idea de último momento y muchos puertos de kits de herramientas de aprendizaje profundo tienen un mantenimiento deficiente, aunque esto está mejorando con el tiempo.

Comenzaremos con algunas instrucciones para instalar el software esperado. paquetes. Luego, echaremos un vistazo rápido a la biblioteca NumPy para Python 3.x. NumPy es fundamental para prácticamente todos los usos científicos de Python y es esencial que sepas cómo trabajar con él en un nivel básico. A continuación, presentaré Ciencia ficción. Este también es un conjunto de herramientas necesario para la ciencia, pero aquí solo necesitaremos una pequeña porción. Finalmente, hablaré un poco sobre el kit de herramientas Scikit-Learn. abreviado aquí como sklearn. Este valioso conjunto de herramientas implementa muchas de las Modelos tradicionales de aprendizaje automático.

A lo largo del libro, a menudo usaré ejemplos en ejecución para ilustrar conceptos. Todos los fragmentos de código asumen que se ha ejecutado la siguiente línea:

---

```
importar numpy como np
```

---

Además, en algunos lugares, el código hará referencia al resultado de un fragmento que apareció anteriormente en este capítulo. Los ejemplos de código son breves, por lo que seguir uno al siguiente no debería ser engorroso. Recomiendo dejar una única sesión de Python en ejecución mientras trabajas en un capítulo, aunque esto no es obligatorio.

## Instalación de los kits de herramientas

El objetivo final de esta sección es tener instalados los siguientes kits de herramientas con al menos el número de versión indicado:

- Python 3.8.5 • NumPy
- 1.17.4 • SciPy 1.4.1 •
- Matplotlib 3.1.2 •
- Scikit-Learn (sklearn) 0.23.2

Es casi seguro que las versiones posteriores a éstas también funcionarán.

Echemos un vistazo rápido a cómo podemos instalar cada uno de estos kits de herramientas en el Principales sistemas operativos.

### linux

Para lo siguiente, el símbolo \$ representa la línea de comando, mientras que >>> es el símbolo de Python.

Una nueva instalación del escritorio Ubuntu 20.04 nos proporciona Python 3.8.5 de forma gratuita. Usa el código

---

```
$ gato /etc/os-liberación
```

---

para verificar la versión de su sistema operativo y usar python3 para ejecutar Python, ya que solo Python inicia el antiguo Python 2.7.

Estos comandos instalan NumPy, SciPy, Matplotlib y sklearn:

---

```
$ sudo apt-get install python3-pip $ sudo apt-
get install python3-numpy $ sudo apt-get install
python3-scipy $ sudo pip3 install matplotlib $ sudo
pip3 install scikit-learn
```

---

Pruebe la instalación iniciando Python 3 e importando cada módulo: numpy, scipy y sklearn. Luego imprima la cadena `__version__` para asegurarse de que cumpla o supere las versiones enumeradas anteriormente. Por ejemplo, consulte el siguiente código.

---

```
>>> importar números; numpy.__version__
'1.17.4'
>>> importar scipy; scipy.__version__
'1.4.1'
>>> importar matplotlib; matplotlib.__version__
'3.1.2'
>>> importar sklearn; sklearn.__version__
'0.23.2'
```

---

## Mac OS

Para instalar Python 3.x para Macintosh, vaya a <https://www.python.org/> y, en Descargas , elija Mac OS X. Luego seleccione la última versión estable de Python 3. Al momento de escribir este artículo, es 3.9.2. Cuando se complete la descarga, ejecute el instalador para configurar Python 3.9.2.

Después de la instalación, abra una ventana de terminal y verifique la instalación con lo siguiente:

---

```
$ python3 --versión
Python 3.9.2
```

---

Suponiendo que Python 3 se haya instalado correctamente, ahora podemos instalar las bibliotecas. usando la ventana de terminal y pip3, que el instalador configuró para nosotros:

---

```
$ pip3 instala numpy --usuario
$ pip3 instala scipy --usuario
$ pip3 instala matplotlib --usuario $
pip3 instala scikit-learn --usuario
```

---

Y, finalmente, podemos comprobar las versiones de las bibliotecas desde Python 3. Ingrese python3 en la terminal para abrir una consola de Python y luego importe numpy, scipy, matplotlib y sklearn e imprima las cadenas de versión, como hicimos anteriormente, para verificar que cumplan o superen las versiones mínimas.

## ventanas

Para instalar Python 3 y los kits de herramientas para Windows 10, siga los siguientes pasos:

1. Vaya a <https://www.python.org/> y haga clic en Descargas y Windows.
2. En la parte inferior de la página, seleccione el instalador ejecutable x86-64.
3. Ejecute el instalador y elija las opciones predeterminadas.
4. Seleccione Instalar para todos los usuarios y agregue Python a la RUTA de Windows.  
Esto es importante.

Cuando finalice el instalador, Python estará disponible desde el símbolo del sistema porque le dijimos al instalador que agregue Python al entorno PATH .

variable. Por lo tanto, abra el símbolo del sistema (WINDOWS-R, cmd) e ingrese python. Si todo va bien, aparecerá el mensaje de inicio de Python y verá un >>> mensaje interactivo. En el momento de escribir este artículo, la versión instalada era 3.8.2. Tenga en cuenta que para salir de Python en Windows, use CTRL-Z, no CTRL-D.

El instalador de Python también tuvo la cortesía de instalar pip . Podemos usarlo directamente desde el símbolo del sistema de Windows para instalar las bibliotecas que necesitemos. Cuando se le solicite, ingrese las siguientes líneas para instalar las bibliotecas NumPy, SciPy, Matplotlib y sklearn :

---

```
> pip instalar numpy >
pip instalar scipy > pip
instalar matplotlib > pip
instalar sklearn
```

---

Para mí, esto instaló NumPy 1.18.1, SciPy 1.4.1, Matplotlib 3.2.1 y sklearn 0.22.2, que cumplen con las versiones mínimas anteriores, por lo que todo está bien.

Para probar cosas, inicie Python desde el símbolo del sistema e importe numpy, scipy, matplotlib y sklearn. Los tres deberían cargarse sin errores. Para escribir código Python, instale cualquier editor con el que se sienta cómodo o simplemente use el Bloc de notas.

Con sus kits de herramientas instalados y listos para funcionar, echemos un vistazo rápido a cada biblioteca para familiarizarnos al menos un poco más con ellas. Veremos ejemplos a lo largo del libro, pero te recomiendo que mires la documentación sugerida. Vale la pena.

## NumPy

Instalamos NumPy en la sección anterior. Ahora presentaré algunos conceptos y manipulaciones básicos de NumPy. Hay un tutorial completo disponible en línea en <https://docs.scipy.org/doc/numpy/user/quickstart.html>.

Inicie Python. Luego intente lo siguiente cuando se le indique:

---

```
>>> importar numpy como
np >>> np.__version__
'1.16.2'
```

---

La primera línea carga NumPy y configura un nombre de acceso directo para él, np. No es necesario utilizar el nombre del acceso directo, pero hacerlo es casi universal. Asumiremos np en el futuro. La segunda línea muestra la versión. Debería ser al menos lo que se muestra arriba.

## Definición de

matrices NumPy opera en matrices y es bastante bueno para convertir listas en matrices. Piense en el tipo de matrices que se encuentran en un lenguaje como C o Java. NumPy proporciona una ventaja porque, aunque Python es elegante, es demasiado lento para usos científicos al simular matrices con listas. Las matrices reales son mucho más rápidas. A continuación se muestra un ejemplo que define una matriz a partir de una lista y luego examina algunas de sus propiedades:

---

```
>>> a = np.matriz([1,2,3,4])
>>> un
      matriz([1, 2, 3, 4]) >>>
un.tamaño
      4
>>> una.forma
      (4,)
>>> una.dtype
      dtype('int64')
```

---

Este ejemplo define una lista de cuatro elementos y luego la pasa a `np.array` para convertirla en una matriz NumPy. Las propiedades básicas de una matriz incluyen el tamaño y la forma. El tamaño es de cuatro elementos. La forma también es cuatro, como una tupla, lo que muestra que `a` es un vector, una matriz unidimensional (1D). La forma es cuatro porque la matriz `a` tiene cuatro elementos. Si `a` fuera bidimensional (2D), la forma tendría dos valores, uno para cada eje de la matriz. Vea el siguiente ejemplo, donde la forma de `b` nos dice que `b` tiene dos filas y cuatro columnas:

---

```
>>> b = np.array([[1,2,3,4],[5,6,7,8]]) >>> imprimir(b) [[1
2 3 4] [5 6 7 8]]
>>> b.forma
      (2, 4)
```

---

## Tipos de

datos Los tipos de datos numéricos de Python vienen en dos tipos: enteros de tamaño arbitrario (pruebe con `2**1000`) o números de punto flotante. NumPy, sin embargo, permite matrices de muchos tipos diferentes. En esencia, NumPy está implementado en C, por lo que admite el mismo conjunto de tipos de datos que admite C. El ejemplo anterior muestra que la función `np.array` tomó la lista dada `y`, dado que cada elemento de la lista era un número entero, creó una matriz donde cada elemento era un entero de 64 bits con signo. La Tabla 1-1 tiene los tipos de datos con los que trabaja NumPy; Podemos dejar que NumPy elija el tipo de datos por nosotros, o podemos especificarlo explícitamente.

Tabla 1-1: Nombres de tipos de datos NumPy, equivalentes de C y rango

Nombre NumPy	Tipo C equivalente	Rango
flotador64	doble	$\pm[2,225 \times 10^{-308}, 1,798 \times 10308]$
flotador32	flotar	$\pm[1,175 \times 1038, 3,403 \times 1038]$
int64	largo largo	$[-263, 263-1]$
uint64	sin firmar largo largo	$[0, 264-1]$
int32	largo	$[-231, 231-1]$
uint32	largo	$[0, 231-1]$
uint8	carácter sin firma	$[0, 255 = 2^8 - 1]$

Veamos algunos ejemplos de matrices con tipos de datos específicos:

---

```
>>> a = np.array([1,2,3,4], dtype="uint8")
>>> tipo.d
tipod('uint8')
>>> a = np.array([1,2,3,4], dtype="int16")
>>> a = np.array([1,2,3,4], dtype="uint32")
>>> b = np.matriz([1,2,3,4.0])
>>> tipo.b.d
tipod('float64')
>>> b = np.array([1,2,3,4.0], dtype="float32")
>>> c = np.array([111,222,333,444], dtype="uint8")
>>> c
matriz([111, 222, 77, 188], tipod=uint8)
```

---

Los ejemplos con matriz a usan tipos enteros y los ejemplos con matriz b usan tipos de punto flotante. Observe que el primer ejemplo b por defecto es un flotante de 64 bits. NumPy hizo esto porque uno de los elementos de la lista de entrada era un flotar (4.0).

El último ejemplo que define la matriz c parece ser un error. Pero no lo es. NumPy no nos advierte si el tipo de datos solicitado no puede contener los valores dados. Aquí, tenemos un entero de 8 bits que sólo puede contener valores en el rango [0, 255]. El Los dos primeros, 111 y 222, encajan, pero los dos últimos, 333 y 444, son demasiado grandes. NumPy mantuvo silenciosamente solo los 8 bits más bajos de estos valores, que corresponden a 77 y 188, respectivamente. La lección es que NumPy espera que sepas lo que estás haciendo con respecto a los tipos de datos. Por lo general, esto no es un problema, pero es algo a tener en cuenta.

## Matrices 2D

Si una lista se convierte en un vector 1D, podríamos sospechar que una lista de listas se convertiría en un vector 1D. en una matriz 2D. Tendríamos razón:

---

```
>>> d = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> d.forma
(3, 3)
```

```
>>> d.tamaño
9
>>> d
matriz([[1, 2, 3], [4, 5,
6], [7, 8, 9]])
```

---

Vemos que una lista de tres sublistas está asignada a una matriz de  $3 \times 3$  (una matriz). Los subíndices en las matrices NumPy cuentan desde cero, por lo que hacer referencia a  $d[1,2]$  arriba devuelve 6.

#### Ceros y unos Dos

funciones NumPy particularmente útiles son `np.zeros` y `np.ones`. Ambos definen matrices dadas una forma. El primero inicializa los elementos de la matriz a cero, mientras que el segundo los inicializa a uno. Esta es la forma principal de crear matrices NumPy desde cero:

```
>>> a = np.zeros((3,4), dtype="uint32") >>> a[0,3]
= 42 >>> a[1,1] = 66

>>> un
matriz([[ 0, 0, 0, 42], [ 0, 66, 0,
0], [ 0, 0, 0, 0]],

dtype=uint32) >>> b = 11*np.ones(( 3,1))
>>> b matriz([[11.], [11.], [11.]])
```

---

El primer argumento es una tupla que indica el tamaño de cada dimensión. Si pasamos un escalar, la matriz resultante es un vector 1D. Veamos la definición de `b`. Aquí, multiplicamos la matriz de  $3 \times 1$  por un escalar (11). Esto hace que cada elemento de la matriz, que se inicializó en 1,0, se multiplique por 11.

#### Indexación avanzada

Vimos indexación de matrices simple en los ejemplos anteriores, donde indexamos con un solo valor. NumPy admite una indexación de matrices más sofisticada. Un tipo que usaremos con frecuencia es un índice único que devuelve un subarreglo completo. He aquí un ejemplo:

```
>>> a = np.arange(12).reshape((3,4))
>>> un
matriz([[ 0, 1, 2, 3], [ 4, 5, 6, 7],
[ 8, 9, 10, 11]]) >>>
a[1]
```

```

matriz([4, 5, 6, 7])
>>> un[1] = [44,55,66,77]
>>> un
matriz([[ 0, 1, 2, 3], [44,
      55, 66, 77], [8, 9,
      10, 11]])

```

---

Este ejemplo presenta np.arange, que es el equivalente NumPy de la función de rango de Python . Observe el uso del método de remodelación para cambiar el vector de 12 elementos a una matriz de  $3 \times 4$ . Además, observe que a[1] devuelve el subarreglo completo, comenzando con el primer índice de la primera dimensión. Esta sintaxis es la abreviatura de a[1,: ] donde : significa todos los elementos de la dimensión dada. Esta abreviatura también funciona para tareas, como muestra la siguiente línea.

La misma sintaxis para indexar sectores de una lista de Python funciona con NumPy. Así es como se ve si continuamos con el ejemplo anterior:

```

>>>
una[:2] matriz([[ 0, 1, 2,
      3], [44, 55, 66,
      77]]) >>>
una[:,:] matriz([[ 0, 1, 2,
      3], [44, 55, 66,
      77]]) >>>
una matriz[:,2,:3]
      ([[ 0, 1, 2], [44,
      55, 66]]) >>> b =
np.arange(12) >>> b matriz([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
      11]) >>>
b[:-2] matriz ([ 0, 2, 4, 6, 8, 10])
>>> b[:-3]
matriz([0, 3, 6, 9])
>>> b[:-1]
matriz([11 , 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

```

---

Vemos que a[:2] devuelve las dos primeras filas con un implícito : para la segunda dimensión, como muestra la siguiente línea. Con nuestro tercer comando, obtenemos un subarreglo en dos dimensiones tomando las primeras dos filas y las primeras tres columnas con a[:,2,:3]. Los ejemplos con b muestran cómo extraer uno de cada dos o cada tres elementos. El último ejemplo es particularmente útil: utiliza un incremento negativo para invertir la dimensión. El incremento es -1 para invertir todos los valores. Si fuera -2, obtendríamos todos los demás elementos de b en orden inverso.

NumPy usa : para indicar todos los elementos a lo largo de una dimensión específica. También permite ... (puntos suspensivos) como abreviatura de "tantos :s como sean necesarios". Por ejemplo, definamos una matriz tridimensional (3D):

---

```
>>> a = np.arange(24).reshape((4,3,2))
>>> un
matriz ([[[ 0,  1],
           [ 2,  3],
           [ 4,  5]],
          [[ 6,  7],
           [ 8,  9],
           [10, 11]],
          [[12, 13],
           [14, 15],
           [16, 17]],
          [[18, 19],
           [20, 21],
           [22, 23]]])
```

---

Puedes pensar en la matriz a como una colección de cuatro matrices de  $3 \times 2$ . Para actualizar la segunda de estas matrices, podría utilizar lo siguiente:

---

```
>>> a[1,:,:] = [[11,22],[33,44],[55,66]]
>>> un
matriz([[[ 0,  1], [ 2,
            3], [ 4,
            5]], [[11,
            22], [33,
            44], [55,
            66]], [[12,
            13], [14,
            15], [16,
            17]], [[18,
            19], [20,
            21], [22,
            23]]])
```

---

Aquí, especificamos las dimensiones explícitamente con : y mostramos que NumPy no es exigente: sabe que una lista de listas coincide con la forma esperada del subarray y actualiza el arreglo a en consecuencia. Obtenemos el mismo efecto usando los puntos suspensivos como se ve a continuación.

---

```
>>> a[2,...] = [[99,99],[99,99],[99,99]]
>>> un
matriz([[ 0,  1], [ 2,
                  3], [ 4,
                  5]], [[11,
                  22], [33,
                  44], [55,
                  66]], [[99,
                  99], [99,
                  99], [99,
                  99]], [[18,
                  19], [20,
                  21], [22,
                  23]]])
```

---

Ahora hemos actualizado el tercer subconjunto de  $3 \times 2$ .

Lectura y escritura en disco Los  
arreglos NumPy se pueden escribir y cargar desde el disco usando `np.save` y `np.load`, así:

---

```
>>> a = np.aleatorio.randint(0,5,(3,4))
>>> un
matriz([[4, 2, 1, 3], [4,
                      0, 2, 4], [0, 4,
                      3, 1]]) >>>
np.save("random.npy",a) > >>
b = np.carga("aleatorio.npy")
>>> b
matriz([[4, 2, 1, 3], [4,
                      0, 2, 4], [0, 4,
                      3, 1]])
```

---

Aquí, estamos usando `np.random.randint` para crear una matriz aleatoria de números enteros de  $3 \times 4$  con valores en el rango de 0 a 5. NumPy tiene bibliotecas extensas para números aleatorios. Escribimos la matriz `a` en el disco como `random.npy`. La extensión `.npy` es necesaria y se agregaría si no la proporcionamos. Luego cargamos la matriz desde el disco usando `np.load`.

Encontraremos otras funciones de NumPy a lo largo del libro. Los explicaré cuando los presenten por primera vez. Pasemos ahora a un vistazo rápido a la biblioteca SciPy.

SciPy agrega una gran cantidad de funciones a Python. Utiliza NumPy bajo el capó, por lo que los dos a menudo se instalan juntos. Un tutorial completo está disponible aquí: <https://docs.scipy.org/doc/scipy/reference/tutorial/index.html>.

En este libro, nos centraremos en las funciones del módulo `scipy.stats`.

Inicie Python y pruebe lo siguiente:

---

```
>>> importar
scipy >>>
scipy.__version__ '1.2.1'
```

---

Esto carga el módulo SciPy y verifica que el número de versión esté en al menos lo que debería ser. Cualquier versión posterior de SciPy debería funcionar bien.

Como prueba rápida, intentemos lo siguiente:

---

```
>>> de scipy.stats importar ttest_ind >>>
a = np.random.normal(0,1,1000) >>>
b = np.random.normal(0,0.5,1000) >>>
c = np.aleatorio.normal(0.1,1,1000) >>>
ttest_ind(a,b)
Ttest_indResult(estadística=-0.027161815649563964, pvalue=0.9783333836992686)
>>> ttest_ind(a,c)
Ttest_indResult(estadística=-2.295584443456226, valorp=0.021802794508002675)
```

---

Primero, cargamos NumPy y luego la función `ttest_ind` del módulo de estadísticas de SciPy. Esta función toma dos conjuntos de datos, digamos puntuaciones de exámenes de dos clases, y plantea la pregunta: ¿estos conjuntos de datos tienen el mismo valor promedio? O, más precisamente, pregunta: ¿hasta qué punto podemos creer que el mismo proceso generó estos dos conjuntos de datos? La prueba t es un método clásico para responder a esta pregunta. Una forma de evaluar su resultado es observar el valor p. Puede pensar en un valor p como la probabilidad de que los dos conjuntos tuvieran la diferencia medida en el valor promedio si provinieran del mismo proceso de generación. Una probabilidad cercana a 1 significa que tenemos mucha confianza en que los dos conjuntos provienen del mismo proceso.

Las variables a, b, c son matrices 1D donde los valores de la matriz (aquí 1000) se extraen de curvas gaussianas, también llamadas curvas normales.

Llegaremos a esto más adelante, pero por ahora, sepá que los números se extraen de una curva de campana donde es más probable que se seleccionen los valores cerca del medio que los que están cerca de los bordes. Los primeros dos argumentos a favor de la normalidad son el valor promedio y la desviación estándar, una medida de qué tan extendida está la curva de campana: cuanto mayor es la desviación estándar, más plana y ancha es la curva.

Para este ejemplo, esperaríamos que a y b fueran muy similares, ya que ambos tienen un valor promedio de 0,0, aunque tienen formas de curva de campana ligeramente diferentes. Sin embargo, c tiene un valor promedio de 0,1. Esperamos que la prueba t detecte esto y nos diga que debemos tener cuidado al creer que a y c fueron generados por el mismo proceso.

La salida de la función `ttest_ind` enumera el valor p (`pvalue`). Y, como esperábamos, comparar a y b arroja un valor p de 0,98, lo que significa que la probabilidad de que veamos la diferencia entre los promedios de estos dos conjuntos de datos, dado que provienen del mismo proceso de generación, es de aproximadamente 98. por ciento. Sin embargo, cuando comparamos a y c, obtenemos un valor p del 2,7 por ciento (0,027). Esto significa que hay alrededor de un 3 por ciento de posibilidades de que veamos la diferencia entre a y c si fueran generados por el mismo proceso. Por lo tanto, concluimos que a y c provienen de procesos diferentes. Afirmamos, entonces, que la diferencia entre estos dos conjuntos de datos es estadísticamente significativa.

Históricamente, los valores de p inferiores a 0,05 se han considerado estadísticamente significativos. Sin embargo, este umbral es arbitrario, y la experiencia reciente en la replicación de experimentos, especialmente en las ciencias blandas, ha llevado a exigir un umbral más estricto. Usar un valor p de 0,05 significa que te equivocarás aproximadamente 1 vez entre 20 ( $1/20 = 0,05$ ), lo cual es un umbral demasiado generoso. Dicho esto, un valor p cercano a 0,05 sugiere que algo está sucediendo y que se justifica realizar más investigaciones (y un conjunto de datos más amplio).

## Matplotlib

Usaremos Matplotlib para generar gráficos. Verifiquemos aquí sus capacidades de trazado 2D y 3D. Primero, un ejemplo 2D simple:

---

```
>>> importar numpy
como np >>> importar matplotlib.pyplot
como plt >>> x =
np.random.random(100) >>> plt.plot(x) >>> plt.show()
```

---

Este ejemplo carga NumPy, con el que Matplotlib funciona mejor, y genera un vector, x, de 100 valores aleatorios, [0, 1], la salida de `np.random.random`. Luego usamos `plt.plot` para trazar el vector y `plt.show` para mostrarlo. La salida de Matplotlib es interactiva. Juegue con la trama para familiarizarse con el uso de la ventana de trama. Por ejemplo, la Figura 1-1 muestra cómo se ve la ventana de trazado en Linux. Como la gráfica es aleatoria, verás una secuencia diferente de valores, pero los controles en la ventana serán los mismos.

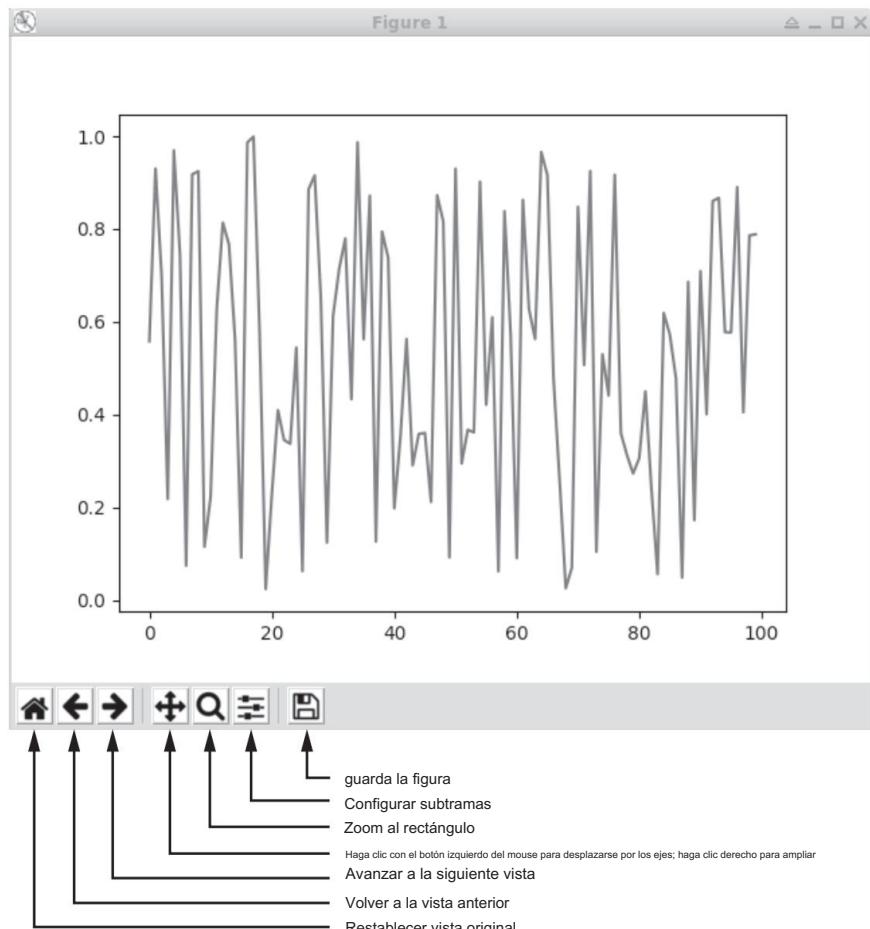


Figura 1-1: Ejemplo de ventana de trazado de Matplotlib

Para 3D, prueba esto:

---

```
>>> desde mpl_toolkits.mplot3d importar Axes3D
>>> importar matplotlib.pyplot como plt
>>> importar numpy
como np >>> x =
np.random.random(20) >>> y
= np.random.random(20 ) >>>
z = np.random.random(20)
>>> fig = plt.figure() >>> ax = fig.add_subplot(111,
proyección='3d') >>>
ax.scatter(x, y,z) >>> plt.mostrar()
```

---

Primero cargamos el kit de herramientas de ejes 3D, Matplotlib y NumPy. Luego, usando NumPy, generamos tres vectores aleatorios, [0, 1). Estos son nuestros puntos 3D.

Usando plt.figure y fig.add\_subplot, configuramos una proyección 3D. El 111 es una abreviatura que le dice a Matplotlib que queremos una cuadrícula de  $1 \times 1$  y que la gráfica actual debe ir en el índice 1 de esa cuadrícula. Entonces, 111 significa una sola trama.

La palabra clave de proyección prepara la trama para 3D. Finalmente se realiza el diagrama de dispersión, ax.scatter, y se muestra, plt.show. Al igual que con el gráfico 2D, el gráfico 3D es interactivo. Agarre y mantenga presionado el mouse para rotar la trama.

## Scikit-Aprende

El objetivo de este libro es cubrir las matemáticas del aprendizaje profundo, no la implementación del aprendizaje profundo. Sin embargo, de vez en cuando, será útil observar uno o dos modelos simples de redes neuronales. En esos casos, usaremos sklearn, en particular, la clase MLPClassifier . Además, sklearn contiene algunas herramientas útiles para evaluar el rendimiento de un modelo y para la visualización de datos de alta dimensión.

Como ejemplo rápido, construyamos una red neuronal simple para clasificar pequeñas imágenes en escala de grises de  $8 \times 8$  píxeles de dígitos escritos a mano. Este conjunto de datos está integrado en sklearn. Aquí está el código para el ejemplo:

---

```
importar numpy
como np desde sklearn.datasets importar
load_digits desde sklearn.neural_network importar MLPClassifier

d = load_digits() dígitos
= d["datos"] etiquetas
= d["objetivo"]

norte = 200

idx = np.argsort(np.random.random(len(etiquetas)))
x_test, y_test = dígitos[idx[:N]], etiquetas[idx[:N]] x_train,
y_train = dígitos[idx[N:]], etiquetas[idx[N:]]

clf = MLPClassifier(hidden_layer_sizes=(128,))
clf.fit(tren_x, tren_y)

puntuación = clf.score(x_test, y_test)
pred = clf.predict(x_test) err =
np.where(y_test != pred)[0]
print("puntuación: ", puntuación)
print("errores:")
print( "real:", y_test[err]) print("predicho:",
pred[err])
```

---

Primero importamos NumPy. Desde sklearn , importamos la función load\_digits para devolver el conjunto de datos de imágenes de dígitos pequeños y la clase MLPClassifier para entrenar una red neuronal tradicional, es decir, un perceptrón multicapa. Luego obtenemos los datos de los dígitos y extraemos las imágenes y sus etiquetas asociadas, 0 . . . 9 . Las imágenes de dígitos se almacenan como vectores de  $8 \times 8 = 64$  elementos que representan la imagen desentrañada de modo que las filas se coloquen de extremo a extremo. El conjunto de datos de dígitos incluye 1797 imágenes, por lo que dígitos es una matriz NumPy 2D con 1797 filas, con 64 columnas por fila, y etiquetas es un vector de etiquetas de 1797 dígitos.

Aleatorizamos el orden de las imágenes, teniendo cuidado de mantener la etiqueta correcta con el dígito correcto y extraemos datos de tren y prueba (`x_train`, `x_test`) y etiquetas (`y_train`, `y_test`). Dejaremos a un lado las primeras imágenes de 200 dígitos para usarlas como datos de prueba y entrenaremos el modelo con las 1597 imágenes restantes. Esto nos deja con aproximadamente 160 imágenes de cada dígito para entrenar y aproximadamente 20 de cada dígito para realizar pruebas.

A continuación, construimos el modelo creando una instancia de `MLPClassifier`. Tomaremos todos los valores predeterminados y especificaremos solo el tamaño de la única capa oculta, que tiene 128 nodos. Los vectores de entrada son 64 elementos, por lo que los duplicamos para la capa oculta. No es necesario especificar explícitamente el tamaño de la capa de salida; sklearn lo deduce de las etiquetas en `y_train`. Entrenar el modelo es una simple llamada a `clf.fit` pasando los vectores de imagen de entrenamiento (`x_train`) y las etiquetas (`y_train`).

El entrenamiento para un pequeño conjunto de datos como este solo llevará unos segundos. Cuando termina, los pesos y sesgos aprendidos están en el modelo (`clf`). Primero obtenemos la puntuación, la precisión general (puntuación) y luego las predicciones de la etiqueta de clase del modelo real en el conjunto de prueba (`pred`). Cualquier error se captura en `err` buscando lugares donde la etiqueta real (`y_test`) no coincide con la predicción. Terminamos mostrando la etiqueta de clase real y la etiqueta prevista para los errores.

Cada vez que ejecutamos este código, obtendremos un orden diferente de los datos de los dígitos, lo que conduce a un tren y un conjunto de prueba diferentes. Además, las redes neuronales se inicializan aleatoriamente antes del entrenamiento. Entonces, obtendremos un resultado diferente cada vez. La primera vez que ejecuté este código, obtuve una puntuación general de 0,97 (97 por ciento) de precisión. Adivinaría una precisión de alrededor del 10 por ciento, por lo que podemos decir que el modelo ha aprendido bastante bien.

## Resumen

En este capítulo, aprendimos cómo configurar nuestro entorno de trabajo. Luego presenté nuestro conjunto de kits de herramientas de Python en un nivel alto y proporcioné sugerencias sobre dónde aprender más. Con el entorno laboral seguro y próspero, el próximo capítulo se sumerge de lleno en la teoría de la probabilidad.



# 2

## PROBABILIDAD



La probabilidad afecta todos los aspectos de nuestras vidas, pero en realidad a todos se nos da bastante mal, como lo demuestran algunos de los ejemplos de este capítulo. Necesitamos estudiar la probabilidad para hacerlo bien. Y debemos hacerlo bien porque el aprendizaje profundo trata ampliamente con ideas de la teoría de la probabilidad. La probabilidad aparece en todas partes, desde los resultados de las redes neuronales hasta la frecuencia con la que aparecen diferentes clases en la naturaleza y las distribuciones utilizadas para inicializar redes profundas.

Este capítulo pretende exponerle a los tipos de ideas relacionadas con la probabilidad y términos que encontrará con frecuencia en el aprendizaje profundo. Comenzaremos con ideas básicas sobre probabilidad e introduciremos la noción de variable aleatoria. Luego pasaremos a las reglas de probabilidad. Estas secciones cubren los conceptos básicos que nos pondrán en condiciones de hablar sobre probabilidades conjuntas y marginales. Encontrará esos términos una y otra vez a medida que explore el aprendizaje profundo. Una vez que comprenda cómo utilizar las probabilidades conjuntas y marginales, le explicaré la primera de las dos reglas de la cadena que se analizan en este libro. El segundo está en el Capítulo 6 sobre cálculo diferencial. Continuaremos nuestro estudio de probabilidad en el Capítulo 3.

## Conceptos básicos

Una probabilidad es un número entre cero y uno que mide la probabilidad de que algo suceda. Si no hay posibilidad de que algo suceda, su probabilidad es cero. Si es absolutamente seguro que sucederá, su probabilidad es uno. Generalmente expresamos las probabilidades de esta manera, aunque en el uso diario, a la gente parece no gustarle decir cosas como: "La probabilidad de que llueva mañana es 0,25". En cambio, decimos: "La probabilidad de que llueva mañana es del 25 por ciento". En el habla cotidiana convertimos la probabilidad fraccionaria a un porcentaje. Haremos lo mismo en este capítulo.

El párrafo anterior utilizó varias palabras asociadas con la probabilidad: probabilidad, azar y certeza. Esto está bien en el uso casual, e incluso en cierto modo en el aprendizaje profundo, pero cuando necesitemos ser explícitos, nos ceñiremos a la probabilidad y la expresaremos numéricamente en el rango de cero a uno, [0, 1]. Los corchetes significan que se incluyen los límites superior e inferior. Si el límite no está incluido en el rango, se utiliza un paréntesis normal. Por ejemplo, la función NumPy `np.random.random()` devuelve un número pseudoaleatorio de punto flotante en el rango [0, 1). Por lo tanto, puede devolver exactamente cero, pero nunca devolverá exactamente uno.

A continuación, presentaré los conceptos fundamentales de espacio muestral, eventos y variables aleatorias. Terminaré con algunos ejemplos de cómo los humanos son malos en probabilidad.

### Espacio muestral y eventos

Dicho de manera sucinta, un espacio muestral es un conjunto discreto o rango continuo que representa todos los resultados posibles de un evento. Un evento es algo que sucede. Por lo general, es el resultado de algún proceso físico, como lanzar una moneda al aire o tirar un dado. Todos los eventos posibles que hemos agrupado son el espacio muestral con el que estamos trabajando. Cada evento es una muestra del espacio muestral y el espacio muestral representa todos los eventos posibles. Veamos algunos ejemplos.

Los posibles resultados de un lanzamiento de moneda son cara (H) o cruz (T); por lo tanto, el espacio muestral para un lanzamiento de moneda es el conjunto {H, T}. El espacio muestral para la tirada de un dado estándar es el conjunto {1, 2, 3, 4, 5, 6} porque, descontando el dado posado en su borde, una de las seis caras del cubo estará encima cuando el dado deja de moverse. Estos son ejemplos de espacios muestrales discretos.

En el aprendizaje profundo, la mayoría de los espacios muestrales son continuos y constan de números de punto flotante, no de números enteros ni de elementos de un conjunto. Por ejemplo, si una característica de entrada a una red neuronal puede tomar cualquier valor en el rango [0, 1], entonces [0, 1] es el espacio muestral para esa característica.

Podemos preguntar sobre la probabilidad de que ocurran ciertos eventos. Por una moneda, Podemos preguntar: ¿cuál es la probabilidad de que la moneda salga cara cuando se lance? Intuitivamente, suponiendo que la moneda no esté ponderada de modo que sea más probable que aparezca una cara que la otra, decimos que la probabilidad de que salga cara es del 50 por ciento. La probabilidad de obtener cara es entonces de 0,5 (50 por ciento como porcentaje). Vemos que la probabilidad de obtener cruz también es 0,5. Finalmente, dado que cara y cruz son los únicos resultados posibles, vemos que la suma de las probabilidades sobre

todos los resultados posibles son  $0,5 + 0,5 = 1,0$ . Las probabilidades siempre suman 1,0 sobre todos los valores posibles del espacio muestral.

¿Cuál es la probabilidad de sacar un cuatro con un dado de seis caras? Nuevamente, no hay razón para favorecer una cara sobre otra, y solo una de las seis caras tiene cuatro puntos, por lo que la probabilidad es una entre seis,  $1/6 \approx 0,166666\ldots$  o alrededor del 17 por ciento.

## Variables aleatorias

Denotemos el resultado de un lanzamiento de moneda por una variable,  $X$ .  $X$  es lo que se llama una variable aleatoria, una variable que toma valores de su espacio muestral con una cierta probabilidad. Como aquí el espacio muestral es discreto,  $X$  es una variable aleatoria discreta, que denotamos con una letra mayúscula. Para la moneda, la probabilidad de que  $X$  sea cara es igual a la probabilidad de que  $X$  sea cruz, ambas 0,5.

Para escribir esto formalmente, usamos

$$P(X = \text{cara}) = P(X = \text{cruz}) = 0,5$$

donde  $P$  se usa universalmente para indicar la probabilidad del evento entre paréntesis para la variable aleatoria especificada. Una variable aleatoria continua es una variable aleatoria de un espacio muestral continuo, denotada con una letra minúscula, como  $x$ . Generalmente hablamos de la probabilidad de que la variable aleatoria esté en algún rango del espacio muestral, no en un número real particular. Por ejemplo, si usamos la función aleatoria NumPy para devolver un valor en  $[0, 1)$ , podemos preguntar: ¿Cuál es la probabilidad de que devuelva un valor en el rango  $[0, 0,25)$ ?

Dado que cualquier número tiene la misma probabilidad de ser devuelto que cualquier otro, decimos que la probabilidad de estar en ese rango es 0,25 o 25 por ciento.

## Los humanos son malos en probabilidad

Nos sumergiremos en las matemáticas de la probabilidad en la siguiente sección. Pero antes de eso, veamos dos ejemplos relacionados con la probabilidad que muestran cuán malos pueden ser los humanos en esto. Ambos ejemplos han dejado perplejos a los expertos, no porque de alguna manera falten expertos, sino porque nuestras intuiciones sobre la probabilidad son a menudo completamente incorrectas, e incluso los expertos son completamente humanos.

## El dilema de Monty Hall

Este problema es uno de mis favoritos, ya que confunde incluso a los matemáticos con títulos avanzados. El dilema está tomado de un viejo programa de juegos estadounidense llamado *Let's Make a Deal*. El presentador original del programa, Monty Hall, seleccionaba a un miembro de la audiencia y le mostraba tres grandes puertas cerradas etiquetadas 1, 2 y 3. Detrás de una de las puertas había un auto nuevo. Detrás de las dos puertas restantes había premios de broma, como una cabra viva.

Se pidió al concursante que eligiera una puerta. Luego, Hall pediría que se abriera una de las puertas que el concursante no eligió, naturalmente una que no tuviera un automóvil detrás. Después de que el público dejara de reírse del premio de broma que había detrás de esa puerta, Hall le preguntaba al concursante si quería

mantener la puerta seleccionada originalmente, o si prefieren cambiar su selección a la puerta restante. El dilema es simplemente ese: ¿mantienen su suposición original o cambian a la puerta restante?

Si quieras pensar en ello un rato, hazlo. Deja el libro, camina, saca un lápiz y un poco de papel, toma notas y luego, cuando tengas una solución (o te rindas), sigue leyendo. . . .

Aquí está la respuesta correcta: cambiar de puerta. Si lo haces, ganarás el coche 2/3 del tiempo. Si no lo haces, sólo ganarás el coche 1/3 de las veces, ya que esa es la probabilidad de seleccionar la puerta correcta inicialmente: una elección correcta entre tres.

Cuando Marilyn vos Savant presentó este problema en su columna de la revista Parade en 1990 y afirmó que la solución correcta es cambiar de puerta, recibió una avalancha de cartas, muchas de ellas de matemáticos, algunos enojados, insistiendo en que estaba equivocada. Ella no lo era. Una forma de ver que tenía razón es utilizar un programa de computadora para simular el juego. No desarrollaremos el código de ninguno aquí, pero no es demasiado difícil. Si escribe uno y lo ejecuta, verá que la probabilidad de ganar al cambiar de puerta converge en 2/3 a medida que aumenta el número de juegos simulados. Sin embargo, también podemos usar el sentido común y las ideas básicas sobre probabilidad para ver la solución.

Primero, si no cambiamos de puerta, sabemos que tenemos 1/3 de probabilidad de ganar el auto. Ahora, considere lo que puede pasar cuando cambiamos de puerta. Si cambiamos de puerta, la única forma de perder es si seleccionamos la puerta correcta en primer lugar. ¿Por qué? Supongamos que inicialmente elegimos una de las puertas del premio de broma. Hall, que sabe muy bien qué puerta está detrás del coche, nunca abrirá la puerta con el coche. Como ya seleccionamos una de las puertas de broma, se ve obligado a elegir la puerta de broma restante y abrirla para nosotros, asegurándose así de que el auto esté detrás de la única puerta restante. Si cambiamos de puerta, ganamos. Como hay dos puertas sin el automóvil, nuestra probabilidad de seleccionar inicialmente la puerta equivocada es 2/3. Sin embargo, acabamos de ver que si elegimos la puerta equivocada inicialmente y cambiamos cuando tenemos la oportunidad, ganaremos el auto. Por lo tanto, tenemos 2/3 de posibilidades de ganar el auto si cambiamos nuestra suposición. La probabilidad de 1/3 de perder al cambiar nuestra suposición inicial es, por supuesto, el caso en el que inicialmente seleccionamos la puerta correcta.

### ¿Cáncer o no?

Este ejemplo se encuentra en varios libros populares sobre probabilidad y estadística (por ejemplo, More Damned Lies and Statistics, de Joel Best [UC Press, 2004], y The Drunkard's Walk, de Leonard Mlodinow [Pantheon, 2008]).

Está basado en un estudio real. La tarea consiste en determinar la probabilidad de que una mujer de unos 40 años tenga cáncer de mama si tiene una mamografía positiva. Tenga en cuenta que las cifras que siguen pueden haber sido precisas cuando se realizó el estudio, pero es posible que no sean válidas ahora. Considérelos sólo como un ejemplo.

Se nos dice lo siguiente:

1. La probabilidad de que una mujer de unos 40 años seleccionada al azar tenga cáncer de mama es del 0,8 por ciento (8 de 1.000).
2. La probabilidad de que una mujer con cáncer de mama tenga una mamografía positiva es del 90 por ciento.
3. La probabilidad de que una mujer sin cáncer de mama tenga una mamografía positiva es del 7 por ciento.

Una mujer llega a la clínica y se hace un examen. La mamografía es positiva. ¿Cuál es la probabilidad, según lo que nos han dicho, de que realmente tenga cáncer de mama?

Del punto 1 anterior, sabemos que si seleccionamos al azar 1000 mujeres de 40 años, 8 de ellas tendrán cáncer de mama (en promedio). Por lo tanto, de esos 8, el 90 por ciento de ellos (n.º 2 arriba) tendrán una mamografía positiva. Esto significa que 7 mujeres con cáncer tendrán una mamografía positiva porque  $8 \times 0,9 = 7,2$ . Esto deja a 992 de las 1.000 originales que no tienen cáncer de mama. Del punto 3 anterior,  $992 \times 0,07 = 69,4$ , por lo que 69 mujeres sin cáncer de mama también tendrán una mamografía positiva, lo que da un total de  $7 + 69 = 76$  mamografías positivas, de las cuales 7 son cáncer real y 69 son falsas.-resultados positivos. Por lo tanto, la probabilidad de que una mamografía positiva indique cáncer es de 7 sobre 76 o  $7/76 = 0,092$ , aproximadamente el 9 por ciento.

La estimación mediana que dieron los médicos que presentaron este problema fue una probabilidad de cáncer de alrededor del 70 por ciento, y más de un tercio da una estimación del 90 por ciento. Las probabilidades son difíciles para los humanos, incluso para aquellos con mucho entrenamiento. El error de los médicos no fue tener en cuenta adecuadamente la probabilidad de que una mujer de unos 40 años seleccionada al azar tuviera cáncer de mama. Veremos en el Capítulo 3 cómo calcular este resultado usando el teorema de Bayes, que sí tiene en cuenta esta probabilidad.

Por ahora, pasemos de la intuición a la formalidad matemática.

## Las reglas de la probabilidad

Comencemos con las reglas básicas de probabilidad. Estas son reglas fundamentales que necesitaremos durante el resto del capítulo y más allá. Aprenderemos sobre la probabilidad de eventos, la regla de la suma de probabilidades y lo que queremos decir con probabilidad condicional. Después de eso, la regla del producto nos permitirá abordar la paradoja del cumpleaños. En la paradoja del cumpleaños, veremos cómo calcular el número mínimo de personas que deben estar juntas en una habitación de modo que la probabilidad de que al menos dos de ellas comparten un cumpleaños supere el 50 por ciento. La respuesta es menos de lo que piensas.

### Probabilidad de un evento

Mencionamos anteriormente que la suma de todas las probabilidades de un espacio muestral es uno. Esto significa que la probabilidad de que ocurra cualquier evento en el espacio muestral es

siempre menor o igual a uno, ya que el evento vino del espacio muestral, y el espacio muestral abarca todos los eventos posibles. Esto implica, para cualquier evento A,

$$0 \leq P(A) \leq 1 \quad (2.1)$$

y, para todos los eventos  $A_i$  en el espacio muestral,

$$\sum_i P(A_i) = 1 \quad (2.2)$$

donde  $\Sigma$  (sigma) significa sumar la expresión de la derecha para cada uno de las  $i$ s. Piense en un bucle for en Python con la expresión de la derecha como cuerpo del bucle.

Si lanzamos un dado de seis caras, intuitivamente (y correctamente) entendemos que la probabilidad de obtener cualquier valor es la misma: una entre seis posibilidades, o  $1/6$ . Por lo tanto, la ecuación 2.1 nos dice que  $P(1)$ , la probabilidad de rodar un uno, está entre cero y uno. Esto es cierto ya que  $0 \leq \frac{1}{6} \leq 1$ . Además- Ademá-s, la ecuación 2.2 nos dice que la suma de las probabilidades de todos los eventos en el espacio muestral debe ser uno. Esto también es válido para el dado de seis caras, ya que  $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6$ .

Si la probabilidad de que ocurra un evento es  $P(A)$ , entonces la probabilidad ese evento A no sucede es

$$P(\bar{A}) = 1 - P(A) \quad (2.3)$$

donde  $P(\bar{A})$  se lee como "no A".  $P(\bar{A})$  se conoce como complemento de A. A veces verás  $P(\bar{A})$  escrito como  $P(\neg A)$  usando  $\neg$ , el símbolo lógico de "no".

La Ecuación 2.3 proviene de la Ecuación 2.1 y la Ecuación 2.2 porque la probabilidad de un evento es menor que uno y la probabilidad de cualquier evento de el espacio muestral que sucede es uno, por lo que la probabilidad de eventos que no son A que ocurra debe ser uno menos la probabilidad de que ocurra el evento A.

Por ejemplo, al lanzar un dado, la probabilidad de obtener un valor en  $[1, 6]$  es uno, pero la probabilidad de obtener un cuatro es  $1/6$ . Entonces, la posibilidad de no sacar un cuatro es la única probabilidad que queda cuando se elimina la posibilidad de sacar un cuatro,

$$P(\bar{A}) = 1 - P(A) = 1 - \frac{1}{6} = \frac{5}{6} = 0,8333\ldots$$

lo que significa que tenemos un 83 por ciento de posibilidades de no sacar un cuatro.

¿Qué pasa si tiramos dos dados y los sumamos? El espacio muestral es el conjunto de números enteros del 2 al 12. Sin embargo, cada suma no es igualmente probable en este. En este caso, una situación que está en el centro del juego de dados de casino, por ejemplo. Nosotros calcular las probabilidades de cada suma enumerando todas las formas en que pueden suceder. Contando las formas en que pueden suceder los eventos y dividiendo por el total número de eventos, podemos determinar la probabilidad. La tabla 2-1 muestra todos los formas posibles de generar cada suma.

Tabla 2-1: Número de combinaciones de dos dados que conducen a Diferentes sumas

Combinaciones de suma		contar probabilidad	
2	1 + 1	1	0.0278
3	1 + 2, 2 + 1	2	0.0556
4	1 + 3, 2 + 2, 3 + 1 1	3	0.0833
5	+ 4, 2 + 3, 3 + 2, 4 + 1 1 +	4	0.1111
6	5, 2 + 4, 3 + 3, 4 + 2, 5 + 1 1 + 6, 2	5	0.1389
7	+ 5, 3 + 4, 4 + 3, 5 + 2, 6 + 1 6 2 + 6, 3 + 5, 4 +		0.1667
8	4, 5 + 3, 6 + 2 5 3 + 6, 4 + 5, 5 + 4, 6 + 3 4 4 + 6,		0.1389
9	5 + 5, 6 + 4 3 5 + 6, 6 + 5 2 6 + 6 1		0.1111
10			0.0833
11			0.0556
12			0.0278
		36	1.0000

En la Tabla 2-1, hay 36 combinaciones posibles de los dos dados. Vemos que la suma más probable es 7, ya que seis combinaciones suman 7. La menos probable son 2 y 12; sólo hay una manera de conseguirlo. Si hay seis maneras de conseguir una suma de 7, entonces la probabilidad de un 7 es “6 de 36”, o  $6/36 \approx 0,1667$ . Volveremos a la tabla 2-1 más adelante en el próximo capítulo cuando analicemos las distribuciones de probabilidad y el teorema de Bayes. La tabla 2-1 ilustra una regla general: si podemos enumerar el espacio muestral, luego podemos calcular las probabilidades de eventos específicos.

Como ejemplo final, si lanzas tres monedas simultáneamente, ¿cuál es el Probabilidad de obtener ninguna cara, una cara, dos caras o tres caras? Nosotros Podemos enumerar los posibles resultados y verlos. Obtenemos lo siguiente:

Las combinaciones de caras cuentan la probabilidad			
0	TTT		0,125
	HTT, THT, TTH	1	0.375
1	HHT, HTH, THH	3	0.375
2 3	HHH	3 1	0,125
		8	1.000

De esta tabla afirmamos que la probabilidad de obtener una o dos caras en tres lanzamientos de moneda ocurre lo mismo: 37,5 por ciento. Probemos esto con un poco de código:

---

```

importar numpy como np
norte = 1000000

M = 3

cabezas = np.ceros(M+1)

para i en el rango (N):
    voltear = np.random.randint(0,2,M)
    h, = np.bincount(volteos, longitud mínima=2)
    cabezas[h] += 1

```

```
prob = caras / N
print("Probabilidades: %s" % np.array2string(prob))
```

---

El código ejecuta 1.000.000 de pruebas (N) simulando el lanzamiento de tres monedas (M). El número de veces que cada prueba termina con 0, 1, 2 o 3 cabezas se almacena en cabezas. Cada prueba selecciona tres valores en [0, 1] (voltea) y cuenta cuántos Aparecen caras (un cero). Usamos np.bincount para esto y descartamos el número de colas. Luego se cuenta el número de caras y la siguiente serie de lanzamientos sucede.

Cuando se completan todas las N simulaciones, convertimos el número de cabezas a probabilidades dividiendo por el número de simulaciones ejecutadas (prob). Finalmente nosotros Imprima las probabilidades correspondientes. Para cero, una, dos o tres caras, una sola ejecución arrojó lo siguiente:

---

Probabilidades: [0,125236, 0,3751, 0,37505, 0,124614]

---

Estas son bastante cercanas a las probabilidades que calculamos anteriormente, por lo que tenemos confianza en que estamos en lo cierto.

#### Regla de la suma

Comenzaremos con una definición: se dice que dos eventos A y B son mutuamente excluyentes si no pueden suceder ambos; sucede lo uno o lo otro. Por ejemplo, al lanzar una moneda al aire sale cara o cruz; no puede ser cara y cruz. Eventos mutuamente excluyentes significa que si ocurre el evento A, el evento B queda excluido y viceversa. viceversa. Además, si las probabilidades de que ocurran dos eventos no tienen ninguna relación, lo que significa que la probabilidad de A no se ve afectada por si B ha sucedido, decimos que los dos eventos son independientes.

La regla de la suma se ocupa de la probabilidad de que ocurra más de un evento mutuamente excluyente. Nos dice la probabilidad de que ocurra cualquiera de los eventos. Por ejemplo, ¿cuál es la probabilidad de sacar un cuatro o un cinco con un morir estándar? Sabemos que la probabilidad de sacar un cuatro es 1/6, al igual que la probabilidad de sacar un cinco. Como los eventos son mutuamente excluyentes, podemos intuir que la probabilidad de obtener un cuatro o un cinco es su suma, ya que cuatro y cinco ya que los resultados son ambas partes del espacio muestral, y uno u otro sucede o no sucede ninguna de las dos cosas. Entonces, obtenemos lo siguiente:

$$P(A \text{ o } B) = P(A \cap B) = P(A) + P(B) \text{ (para eventos mutuamente excluyentes)} \quad (2.4)$$

Aquí "o" significa "o" o "unión". Verás a menudo. Para un troquel estándar, la probabilidad de sacar un cuatro o un cinco es  $\frac{1}{6} + \frac{1}{6} = \frac{1}{3}$ , o alrededor del 33 por ciento. espacio muestral de dos lanzamientos de moneda es {HH, HT, TH, TT}; por lo tanto, esto es la probabilidad de obtener dos caras o dos cruces:

$$P(HH \text{ o } TT) = P(HH) + P(TT) = + \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$

Hay más en la regla de la suma, pero antes de que podamos verlo, debemos considerar la regla del producto.

### Regla del producto

La regla de la suma nos informa sobre la probabilidad de que ocurran los eventos A o B. La regla del producto nos dice la probabilidad de los eventos A y B:

$$P(A \text{ y } B) = P(A \cap B) = P(A)P(B) \quad (2.5)$$

Aquí  $\cap$  significa "y" o "intersección".

Si los eventos A y B son mutuamente excluyentes, inmediatamente veremos que  $P(A \cap B) = 0$  porque si el evento A ocurre con probabilidad  $P(A)$ , entonces la probabilidad del evento B es  $P(B) = 0$ , y su producto es también cero. Lo mismo ocurre si ocurre el evento B; entonces  $P(A) = 0$ .

Por supuesto, no todos los acontecimientos son mutuamente excluyentes. Por ejemplo, supongamos que el 80 por ciento de las personas en el mundo tienen ojos marrones y el 50 por ciento son mujeres. ¿Cuál es la probabilidad de que una persona seleccionada al azar sea una mujer con ojos marrones? Usemos la regla del producto,

$$P(\text{mujer, ojos marrones}) = P(\text{mujer})P(\text{ojos marrones}) = 0,5(0,8) = 0,4$$

para ver que hay un 40 por ciento de posibilidades de que una persona seleccionada al azar sea una mujer de ojos marrones.

La regla del producto tiene sentido si lo pensamos un poco. Calcular la fracción de personas, que es la probabilidad, que son mujeres no cambiará la fracción de mujeres que tienen ojos marrones. Un evento, ser mujer, no tiene ningún impacto en el otro evento, tener ojos marrones.

La regla del producto no se limita a sólo dos eventos. Considera lo siguiente. Segundo las compañías de seguros, la probabilidad de que le caiga un rayo en un año determinado, si vive en los EE. UU., es de aproximadamente 1/1.222.000, o 0,0000082 por ciento. ¿Cuál es la probabilidad de ser una mujer de ojos marrones y ser alcanzado por un rayo en un año determinado, suponiendo que viva en los EE. UU.?

Nuevamente podemos usar la regla del producto:

$$P(\text{mujer, ojos marrones, relámpago}) = P(\text{mujer})P(\text{ojos marrones})P(\text{relámpago})$$

$$= 0,5(0,8)(0,00000082)$$

$$= 0,00000033 = 0,000033\%$$

La población de Estados Unidos es de aproximadamente 331.000.000 de habitantes, de los cuales el 0,000033 por ciento son mujeres de ojos marrones que serán alcanzadas por un rayo este año: 109 personas, según nuestro cálculo anterior. Según el Servicio Meteorológico Nacional de Estados Unidos, cada año unas 270 personas serán alcanzadas por un rayo. Como vimos anteriormente, el 40 por ciento de esas personas serán mujeres de ojos marrones, lo que arroja  $270 (0,4) = 108$ . Por lo tanto, nuestro cálculo es completamente creíble.

### Regla de la suma revisada

Dijimos anteriormente que hay más en la regla de la suma. Veamos ahora lo que nos faltaba arriba.

La ecuación 2.4 nos da la regla de la suma para los eventos A y B mutuamente excluyentes. ¿Qué pasa si los eventos no son mutuamente excluyentes? En ese caso, es necesario modificar la regla de la suma:

$$P(A \text{ o } B) = P(A) + P(B) - P(A \text{ y } B) \quad (2.6)$$

Veamos un ejemplo.

Un arqueólogo ha descubierto un pequeño alijo de 20 monedas antiguas. Observa que 12 de las monedas son romanas y 8 son griegas. También señala que 6 de las monedas romanas y 3 de las griegas son de plata. Las monedas restantes son de bronce. ¿Cuál es la probabilidad de seleccionar una moneda de plata o romana del alijo?

Si creemos que la plata y el romano son mutuamente excluyentes, estaríamos tentados a decir lo siguiente:

$$P(\text{plata o romano}) = P(\text{plata}) + P(\text{romano})$$

$$= \frac{9}{20} + \frac{12}{20} \quad (\text{¡Esto está mal!})$$

Sin embargo, la suma de las dos probabilidades es  $\frac{21}{20} = 1.05$ , y no podemos tener una probabilidad mayor que uno. Algo anda mal.

El problema es que en el escondite hay monedas romanas que están hechas de plata. Los contamos dos veces, una en  $P(\text{plata})$  y otra en  $P(\text{romana})$ . entonces ahora necesitamos restarlos de la suma total. Hay seis de plata.

Monedas romanas. Entonces, la probabilidad de ser una moneda romana de plata es  $P(\text{plata y romana}) = \frac{6}{20}$ . Restando esa parte, vemos que la probabilidad de escoger una moneda de plata o una moneda romana es del 75 por ciento:

$$P(\text{plata o romano}) = P(\text{plata}) + P(\text{romano}) - P(\text{plata y romano})$$

$$\begin{aligned} &= \frac{9}{20} + \frac{12}{20} - \frac{6}{20} \\ &= \frac{15}{20} = 0,75 \end{aligned}$$

Al igual que con la regla de la suma, hay más en la regla del producto y llegaremos a eso en breve. Pero primero, usemos la regla del producto para ver si podemos resolver la paradoja del cumpleaños.

### La paradoja del cumpleaños

En promedio, ¿cuántas personas necesitamos juntas en una habitación para tener una probabilidad superior al 50 por ciento de que dos de ellas comparten el mismo cumpleaños? Este problema se conoce como la paradoja del cumpleaños. Veamos si podemos usar nuestro conocimiento de la regla del producto de probabilidad para ver cuál es la solución.

Ignoraremos los años bisiestos y afirmaremos que un año tiene 365 días. Intuitivamente, vemos que la probabilidad de que personas seleccionadas al azar comparten un cumpleaños es de un día (el cumpleaños compartido) de 365 cumpleaños posibles en un año. El espacio muestral es de 365 días y el cumpleaños compartido es el día en común. Entonces, obtenemos lo siguiente:

$$P(\text{compartir un cumpleaños}) = \frac{1}{365} \approx 0,00274$$

O comparten fecha de nacimiento o no:  $1 - 1/365 = 365/365 - 1/365 = 364/365$ . Entonces obtenemos lo siguiente:

$$P(\text{no comparte cumpleaños}) = 1 - \frac{1}{365} = \frac{364}{365} \approx 0,9973$$

De los 365 días de un año, hay una coincidencia posible, quedando 364 días que no coinciden.

Una probabilidad del 0,3 por ciento de que personas seleccionadas al azar compartan su cumpleaños es bastante baja. Significa que si eliges al azar pares de personas y les preguntas si comparten la misma fecha de nacimiento, en promedio obtendrás tres coincidencias entre mil, algo que no es muy probable.

Para nuestro cálculo, veremos las cosas desde el otro lado. Estamos buscando la cantidad de personas que necesitamos juntas para que la probabilidad de que no haya dos personas que comparten el mismo cumpleaños sea inferior al 50 por ciento.

Conocemos la probabilidad de que dos personas seleccionadas al azar no comparten 365 . Por un cumpleaños:<sup>364</sup> tanto, si seleccionamos dos pares de personas al azar, el la probabilidad de que ambas parejas no comparten un cumpleaños es la siguiente:

$$\begin{aligned} P(\text{ninguno comparte cumpleaños}) &= P(\text{no comparte})P(\text{no comparte}) \\ &= (\overline{364} \overline{365}) (\overline{364} \overline{365}) \\ &= 0,9945 = 99,45\% \end{aligned}$$

Aquí estamos usando la regla del producto. De manera similar, con tres personas, (A, B, C), podemos formar tres parejas diferentes, (A, B), (A, C) y (B, C), por lo que calculamos lo siguiente:

$$P(\text{sin cumpleaños compartido}) = (\overline{364} \overline{365}) (\overline{364} \overline{365}) (\overline{364} \overline{365})$$

Para n comparaciones, aquí está la probabilidad de que ninguna comparta un cumpleaños:

$$P(\text{sin cumpleaños compartido}) = (\overline{364} \overline{365})^n \quad (2.7)$$

Nuestra tarea es encontrar el número mínimo de comparaciones, n, que conduzcan a una probabilidad de que no se comparta el cumpleaños < 50 por ciento, donde n es una función del número de personas en la habitación, m. ¿Por qué menos del 50 por ciento? Porque si encontramos una n que conduce a una probabilidad inferior al 50 por ciento de que no exista

cumpleaños compartido, la probabilidad de que haya un cumpleaños compartido debe ser > 50 por ciento.

Si eliges a tres personas al azar, hay tres pares de personas para verificar si comparten la misma fecha de cumpleaños. Si tienes cuatro personas, hay seis parejas. Entonces, cuanto mayor es el grupo de personas, más parejas hay. ¿Podemos encontrar una regla que relacione el número de personas, m, con el número de pares a comparar, n? Si tenemos eso, podemos encontrar la m más pequeña que conduzca a una n donde la probabilidad de la Ecuación 2.7 es <50 por ciento.

Cuando tenemos un conjunto de m objetos únicos, como personas en una habitación, y seleccionamos pares de ellos, ¿cuántos pares diferentes podemos seleccionar? En otras palabras, ¿cuántas combinaciones de m cosas hay cuando se toman dos a la vez?

La fórmula para calcular el número de combinaciones de m cosas tomadas k a la vez es la siguiente:

$$C(m, k) = \frac{m!}{k!(m - k)!}$$

A veces escucharás que esto se denomina "m elige k", donde, para nosotros, k = 2.

Encontremos el número de comparaciones que necesitamos, n, y usemos el número de combinaciones de cosas tomadas de dos en dos para encontrar una m que conduzca a al menos n comparaciones.

Un bucle sencillo en Python localiza la n que necesitamos:

---

```
para n en el
    rango(300): si ((364/365)**n
        < 0,5):
            print(n) break
```

---

Se nos dice que n = 253. Entonces, necesitamos hacer, en promedio, 253 comparaciones, 253 pares de personas, para tener una probabilidad superior al 50 por ciento de que uno de esos pares comparta el mismo cumpleaños. El paso final es encontrar cuántas combinaciones de m personas tomadas de dos en dos son al menos 253. Un poco de prueba y error de fuerza bruta nos dice esto:

$$\begin{aligned} \binom{23}{2} &= \frac{23!}{2!(23-2)!} \\ &= \frac{23!}{2!(21)!} \\ &= \frac{23(22)}{2} \\ &= 253 \end{aligned}$$

Necesitamos m = 23 personas en promedio para tener una probabilidad superior al 50 por ciento de que al menos dos de ellas coincidan en su cumpleaños. Todo gracias a la regla del producto.

¿Nuestro resultado es confiable o simplemente un juego de manos? Algun código puede decirnos. Primero, Verifiquemos mediante simulación que la probabilidad de elegir aleatoriamente a dos personas que comparten el mismo cumpleaños es del 0,3 por ciento:

---

```

partido = 0
para i en el rango (100000): a
    = np.random.randint(0,364) b =
    np.random.randint(0,364) si (a == b):
        coincide += 1

print("Probabilidad de una coincidencia aleatoria = %0.6f" % (coincidencia/100000.))

```

---

El código simula 100.000 pares aleatorios de personas, donde el número entero aleatorio en [0, 364] representa el cumpleaños de la persona. Si los dos días de cumpleaños aleatorios coinciden, la coincidencia se incrementa. Después de ejecutar todas las simulaciones, imprimimos la probabilidad. Una ejecución de este código produjo lo siguiente, lo que hace creíble nuestra afirmación de una probabilidad del 0,3 por ciento:

---

Probabilidad de una coincidencia aleatoria = 0,003100

---

¿Qué pasa con el número de personas que tienen una probabilidad > 50 por ciento de compartir? ¿Tienes un cumpleaños? Aquí tenemos dos bucles. El primero es sobre el número de personas en la sala (m), y el segundo es sobre el número de simulaciones para esa cantidad de personas en la sala (n). En código, se ve así:

---

```

para m en rango(2,31):
    coincidencias
    = 0 para n en rango(100000):
        coincidencia
        = 0 b = np.random.randint(0,364,m) para
            i en rango(m): para j en
                rango(m ): si (i != j) y
                    (b[i] == b[j]):
                        partido += 1
        si (coincide! = 0):
            coincide += 1
    print("%2d %0.6f" % (m, coincidencias/100000))

```

---

Dejamos que entre 2 y 30 personas. Para cada grupo de m personas, realizamos 100.000 simulaciones. Para cada simulación, elegimos un conjunto de cumpleaños para cada persona en la habitación (b) y luego comparamos a cada persona con todas las demás para ver si hay un cumpleaños coincidente. Si lo hay, incrementamos la coincidencia. Si tuvimos al menos una coincidencia, incrementamos las coincidencias y pasamos a la siguiente simulación. Finalmente, cuando se completan todas las simulaciones para el número actual de personas en la sala, imprimimos la probabilidad de al menos una coincidencia.

Si ejecutamos el código y trazamos el resultado, obtenemos la Figura 2-1, donde la línea discontinua es el 50 por ciento. El primer punto encima de la línea discontinua son 23 personas, exactamente como calculamos.

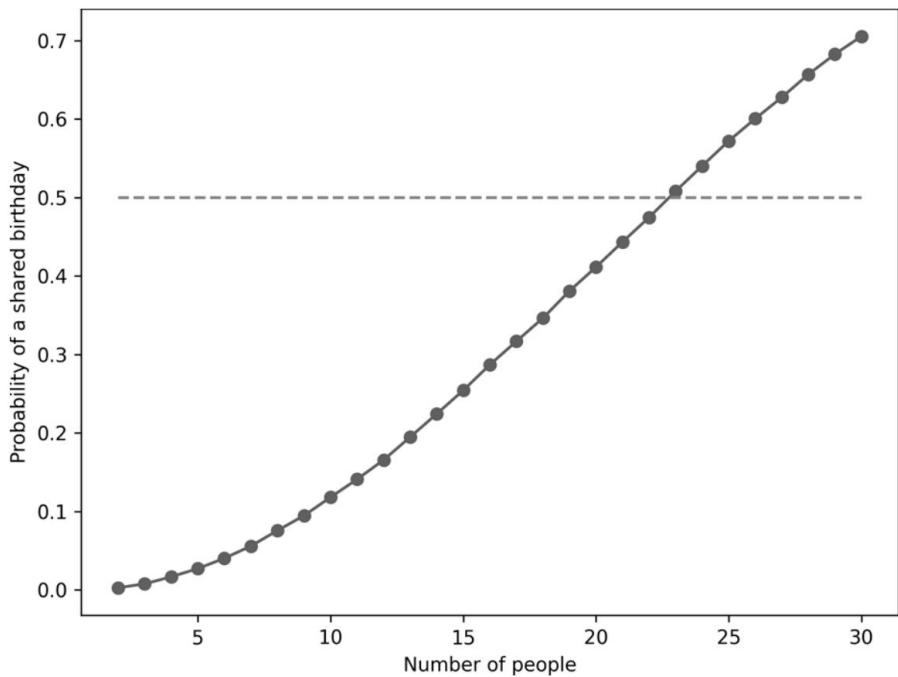


Figura 2-1: La probabilidad de un cumpleaños compartido en función del número de personas  
En una habitación

Siempre es satisfactorio ver que la simulación se alinea con las matemáticas.

#### Probabilidad condicional

Considere una bolsa de 10 canicas: 8 rojas y 2 azules. Sabemos que si sacamos una canica al azar de la bolsa, tenemos una probabilidad de 2 sobre 10, o 20 por ciento, de sacar una canica azul. Digamos que elegimos una canica azul. Después de admirar su bonito tono azul, la volvemos a meter en la bolsa, la agitamos y sacamos otra canica. ¿Cuál es nuestra probabilidad de sacar una canica azul por segunda vez? Nuevamente, hay 2 canicas azules y 10 en total, por lo que sigue siendo el 20 por ciento.

Si el hecho de que ocurrió el evento A (en este caso, coger una canica azul que luego devolvimos a la bolsa) no ha afectado la probabilidad de un evento B futuro, los dos son eventos independientes. Nuestra probabilidad de escoger una canica azul por segunda vez no se ve afectada de ninguna manera por el hecho de que previamente elegimos una canica azul. Lo mismo ocurre con el lanzamiento de una moneda. El hecho de que hayamos salido cara cuatro veces seguidas no tiene nada que ver con la probabilidad de salir cruz en el siguiente lanzamiento, suponiendo que sea una moneda justa, es decir, que no tenga peso en una cara o que tenga dos caras (o dos). -cola).

Ahora, considere un escenario alternativo. Todavía tenemos una bolsa con ocho canicas rojas y dos azules. Escogemos una canica (digamos que esta vez es roja) y como nos gusta el color, nos quedamos con la canica y la dejamos a un lado. Ahora, sacamos otra canica de la bolsa. ¿Cuál es la probabilidad de sacar otra canica roja?

Aquí las cosas han cambiado. Ahora hay nueve canicas y siete de ellas son rojas. Entonces, nuestra probabilidad de escoger una segunda canica roja es ahora 7 de 9, o

78 por ciento. La posibilidad de elegir una canica roja inicialmente era de 8 de 10, es decir, el 80 por ciento. El hecho de que haya ocurrido el evento A, coger una canica roja que luego conservamos, ha alterado la probabilidad de un segundo evento. Los dos eventos ya no son independientes. La probabilidad del segundo evento cambió cuando ocurrió el primer evento. Notacionalmente, escribimos  $P(B|A)$  para indicar la probabilidad del evento B dado que ha ocurrido el evento A. Esta es una probabilidad condicional porque está condicionada a que ocurra el evento A.

Aquí es donde actualizamos la regla del producto. La versión de la Ecuación 2.5 supone que los dos eventos son independientes, como ser mujer y tener ojos marrones. Si tenemos una situación dependiente, la regla pasa a ser

$$P(A \text{ y } B) = P(B|A)P(A) \quad (2.8)$$

lo que significa que la probabilidad de que ocurran dos eventos es el producto de la probabilidad de que uno haya ocurrido el otro y la probabilidad del otro.

Volviendo a nuestro ejemplo de canicas anterior, calculamos que la probabilidad de escoger una canica roja después de haber elegido y conservado una canica roja era 7 de 9, o alrededor del 78 por ciento. Eso es  $P(B|A)$ . Para  $P(A)$ , necesitamos la probabilidad de sacar inicialmente una canica roja, que dijimos que era del 80 por ciento. Por lo tanto, la probabilidad de sacar una canica roja que conservamos, A, y sacar una canica roja en una segunda extracción, B, es del 62 por ciento:

$$P(A \text{ y } B) = P(B|A)P(A)$$

$$\begin{aligned} &= \left( \frac{8}{9} \right) \left( \frac{10}{10} \right) \\ &= 0,6222 \end{aligned}$$

Si dos eventos son mutuamente excluyentes,  $P(B|A) = P(A|B) = 0$ . Si los eventos A y B son independientes, entonces  $P(A|B) = P(A)$  y  $P(B|A) = P(B)$  porque el evento condicional que ocurra o no no tiene influencia en el evento posterior.

Finalmente, tenga en cuenta que normalmente  $P(B|A) \neq P(A|B)$ , y confundir las dos. Las probabilidades adicionales son un error común y a menudo grave. Como veremos en el capítulo 3, algo llamado teorema de Bayes proporciona la relación adecuada entre las probabilidades condicionales. Volveremos a encontrarnos con la probabilidad condicional cuando analicemos la regla de la cadena para la probabilidad.

### Probabilidad total

Si nuestro espacio muestral está separado en regiones disjuntas,  $B_i$  ( $B_1, B_2$ , etc.) de modo que la totalidad del espacio muestral esté cubierta por la colección de  $B_i$ s y los  $B_i$ s no se superpongan, podemos calcular la probabilidad de un evento sobre todas las particiones de la siguiente manera:

$$P_{\text{AG}}(A) = \sum_i P(A|B_i)P(B_i)$$

Aquí  $P(A|B_i)$  es la probabilidad de A dada la partición  $B_i$  y  $P(B_i)$  es la probabilidad de la partición  $B_i$ , que es la cantidad de espacio muestral que  $B_i$  representa. Desde este punto de vista,  $P(A)$  es la probabilidad total de A sobre las particiones,  $B_i$ .

Veamos un ejemplo de cómo utilizar esta ley.

Tienes tres ciudades, Kish, Kesh y Kuara, y sus poblaciones son 2000, 1000 y 3000, respectivamente. Además, los porcentajes de personas con ojos azules en estas ciudades son del 12 por ciento, 3 por ciento y 21 por ciento, respectivamente. Queremos saber la probabilidad de que una persona seleccionada al azar entre las ciudades tenga ojos azules. La población de las ciudades afecta las cosas, ya que la probabilidad de tener ojos azules varía según la ciudad y las ciudades varían en población. Para encontrar  $P(\text{azul})$ , usamos la probabilidad total:

$$\begin{aligned} P(\text{azul}) &= P(\text{azul}|Kish)P(Kish) \\ &\quad + P(\text{azul}|Kesh)P(Kesh) \\ &\quad + P(\text{azul}|Kuara)P(Kuara) \end{aligned}$$

Aquí  $P(\text{azul}|Kish)$  es la probabilidad de tener ojos azules dado que vives en Kish, y  $P(Kish)$  es la probabilidad de vivir en Kish, y así sucesivamente.

Conocemos las cantidades necesarias para encontrar la probabilidad total. La probabilidad de tener ojos azules por ciudad se da arriba, y la probabilidad de vivir en cada ciudad se calcula a partir de su población y la población total de las tres ciudades:

$$P(Kish) = \frac{2000}{6000} = \frac{1}{3}$$

$$P(Kesh) = \frac{1000}{6000} = \frac{1}{6}$$

$$P(Kuara) = \frac{3000}{6000} = \frac{1}{2}$$

Por lo tanto,  $P(\text{azul})$  es

$$P(\text{azul}) = 0,12 \left( \frac{1}{3} \right) + 0,03 \left( \frac{1}{6} \right) + 0,21 \left( \frac{1}{2} \right) = 0,15$$

lo que significa que hay un 15 por ciento de posibilidades de que un habitante seleccionado al azar de las tres ciudades tenga ojos azules. Observe la suma de las probabilidades para seleccionar las ciudades:  $P(Kish) + P(Kesh) + P(Kuara) = 1$ . Este debe ser el caso para la partición del espacio muestral total, todos los habitantes de las ciudades., que quedará cubierto por la partición en ciudades.

## Probabilidad conjunta y marginal

La probabilidad conjunta de dos variables,  $P(X = x, Y = y)$ , es la probabilidad de que la variable aleatoria X tenga el valor x al mismo tiempo que la variable aleatoria

Y es y. Ya hemos visto un ejemplo de probabilidad conjunta. cuando usamos "y" al calcular una probabilidad, estamos calculando una probabilidad conjunta. Una probabilidad conjunta es la probabilidad de que múltiples condiciones sean verdaderas en el al mismo tiempo, que es "y". La probabilidad marginal es la que obtenemos cuando calcular la probabilidad de una o más de esas condiciones sin importarle sobre el valor de los demás; en otras palabras, la probabilidad de que un subconjunto de las variables aleatorias en el "y".

En esta sección, examinaremos las probabilidades conjuntas y marginales usando tablas simples. Luego introduciremos la regla de la cadena para la probabilidad. Esta regla permite Descompongamos una probabilidad conjunta en el producto de probabilidades conjuntas más pequeñas y probabilidades condicionales.

#### Tablas de probabilidad conjunta

Según Color Blind Awareness (<http://www.colourblindawareness.org/>), Aproximadamente 1 de cada 12 hombres y 1 de cada 200 mujeres son daltónicos. La diferencia proviene del hecho de que el gen afectado está en el cromosoma X, Requerir que una mujer herede el gen recesivo tanto de su madre como de su madre. padre. Un hombre sólo necesita heredar el gen de uno de sus padres.

Imaginemos que encuestamos a 1.000 personas. Podemos contar el número de personas. que son hombres y daltónicos, mujeres y daltónicos, hombres y no daltónicos, y mujeres y no daltónicos. Hacemos esto y organizamos los datos en un tabla así:

	Daltónico	No daltónico	
Hombre	42	456	498
Mujer	345	499	502
	955		1000

Este tipo de tablas se conocen como tablas de contingencia. Los datos contabilizados se encuentran en el centrar la porción numérica de  $2 \times 2$  de la tabla. La columna de la derecha es la suma entre las filas y la última fila es la suma de las columnas. La suma de la última fila o columna está en la última celda y, por necesidad, suma el 1.000 personas que encuestamos.

Podemos convertir la tabla de contingencia en una tabla de probabilidades dividiendo cada celda entre 1.000, el número de personas encuestadas. Hacer esto nos da la siguiente:

	Daltónico	No daltónico	
Masculino	0,042	0,456	0,498
Femenino	0,003	0,499	0,502
	0,045	0,955	1.000

La tabla es ahora una tabla de probabilidad conjunta. Con él podemos buscar el probabilidad de ser hombre y daltónico. Notacionalmente escribimos

$$P(\text{sexo} = \text{masculino}, \text{daltónico} = \text{sí}) = 0,042$$

y de la misma manera vemos que

$$P(\text{sexo} = \text{mujer}, \text{daltónico} = \text{no}) = 0,499$$

Usando la tabla de probabilidad conjunta, podemos predecir lo que esperaríamos medir dada una muestra aleatoria de personas. Por ejemplo, si tenemos una muestra de 20.000 personas, entonces, según nuestra tabla, esperaremos encontrar alrededor de 20.000 (0,042) = 840 hombres daltónicos y alrededor de 20.000 (0,003) = 60 daltónicos. mujer.

¿Y si quisieramos saber la probabilidad de ser daltónico independientemente del sexo? Para eso, sumamos las probabilidades a lo largo de la columna de daltónicos y vemos que hay un 4,5 por ciento de posibilidades de que una persona seleccionada al azar sea daltónica. Del mismo modo, la suma a lo largo de la fila nos da una probabilidad estimada de ser mujer del 50,2 por ciento. Si debemos tener en cuenta que nuestra mesa se construyó a partir de una muestra de sólo 1.000 personas. Se podría suponer que si en lugar de ello hubiéramos muestreado a 100.000 personas, nuestra división entre hombres y mujeres sería más cercana al 50/50, y estaría en lo cierto.

Calcular la probabilidad de ser daltónico o mujer del porro  
La tabla de probabilidad calcula una probabilidad marginal. En el primer caso, sumamos a lo largo de la columna para eliminar el efecto del sexo, mientras que en el segundo caso, sumamos a lo largo de la fila para eliminar el efecto del daltonismo.

Matemáticamente, obtenemos las probabilidades marginales sumando las variables que no queremos. Si tenemos una tabla de probabilidad conjunta para dos variables, como en el ejemplo anterior, obtenemos las probabilidades marginales sumando:

$$P(X = x) = \sum_i P(X = x, Y = y_i)$$

$$P(Y = y) = \sum_i P(X = x_i, Y = y)$$

Usando la tabla anterior, podemos escribir

$$P(Y = \text{daltónico}) = P(X = \text{hombre}, Y = \text{daltónico})$$

$$+ P(X = \text{mujer}, Y = \text{daltónico})$$

donde sumamos el sexo para eliminar su efecto. Ahora exploremos otra tabla, una con tres variables.

En algún momento de la noche del 14 de abril de 1912, el RMS Titanic se hundió en el Atlántico Norte en su viaje inaugural desde Inglaterra a la ciudad de Nueva York. Con base en una muestra de 887 personas que estaban a bordo del Titanic, podemos generar la tabla 2-2 que muestra la probabilidad conjunta de tres variables: supervivencia, sexo y clase de cabina.

Tabla 2-2: Tabla de probabilidad conjunta del Titanic  
Pasajeros

		Cabaña1	Cabaña2	Cabaña3
Muerto	Masculino	0,087	0,103	0,334
	Mujer	0,003	0,007	0,081
Vivo	Masculino	0,051	0,019	0,053
	Femenino	0,103	0,079	0,081

Usemos la Tabla 2-2 para calcular algunas probabilidades. Nota, usaremos el valores en la Tabla 2-2, que tienen una precisión de tres decimales. Como resultado, el Los números generales estarán ligeramente alejados de las probabilidades que calcularíamos. de los conteos, pero hacer esto hace que el vínculo entre la tabla y el ecuaciones más concretas.

Primero, podemos leer directamente de la tabla los trillizos específicos de supervivientes, sexo y clase de cabina. He aquí un ejemplo:

$$P(\text{muerto, hombre, cabina3}) = 0,334$$

Esto significa la probabilidad de que un pasajero seleccionado al azar sea un hombre. quienes estaban en una cabina de tercera clase y no sobrevivieron es el 33 por ciento. Qué pasa ¿Hombres en primera clase? Eso también está en la tabla:

$$P(\text{muerto, hombre, cabina1}) = 0,087$$

Esto significa que un pasajero seleccionado tiene un 9 por ciento de posibilidades de ser un hombre. en primera clase que murió. Podemos ver que las diferencias de clase, en los camarotes y en la sociedad, importaban bastante.

Usemos la tabla para calcular algunas otras probabilidades conjuntas y marginales. Primero, ¿cuál es la probabilidad de no sobrevivir? Para encontrarlo, necesitamos sumar sobre sexo y cabina:

$$\begin{aligned} P(\text{muerto}) &= P(\text{muerto, M, 1}) + P(\text{muerto, M, 2}) + P(\text{muerto, M, 3}) \\ &\quad + P(\text{muerto, F, 1}) + P(\text{muerto, F, 2}) + P(\text{muerto, F, 3}) \\ &= 0,087 + 0,103 + 0,334 + 0,003 + 0,007 + 0,081 \\ &= 0,615 \end{aligned} \tag{2.9}$$

Aquí hemos introducido una notación abreviada para hombre/mujer (M/F) y clase de cabina (1, 2, 3).

Calculemos la probabilidad de no sobrevivir dado que el pasajero estaba macho,  $P(\text{muerto}|M)$ . Para hacer esto, volvemos a la ecuación 2.8, recordando que el "y" implica una probabilidad conjunta. Reescribimos la ecuación 2.8 para resolver para  $P(B|A)$ :

$$P(B|A) = \frac{P(A, B)}{\text{PENSILVANIA}}$$

A veces, esto se utiliza para definir la probabilidad condicional en primer lugar. Tenga en cuenta que  $P(A, B)$  significa  $P(A \text{ y } B)$ : ambas son probabilidades conjuntas. Usando esta forma, la probabilidad de no sobrevivir dado que el pasajero es hombre es

$$P(\text{muerto}|M) = \frac{P(\text{muerto}, M)}{P(M)}$$

donde  $P(\text{muerto}, M)$  es la probabilidad conjunta de estar muerto y ser hombre, y  $P(M)$  es la probabilidad de ser hombre.

Debemos tener cuidado al pensar en las probabilidades.  $P(\text{muerto}, M)$  no es la probabilidad de no sobrevivir si el pasajero es hombre. En cambio, es la probabilidad de que un pasajero seleccionado al azar sea un hombre que no sobrevivió. Lo que queremos es  $P(\text{muerto}|M)$ , que es la probabilidad de no sobrevivir dado que el pasajero era hombre.

Para obtener  $P(\text{muerto}, M)$ , necesitamos sumar la clase de cabina:

$$\begin{aligned} P(\text{muerto}, M) &= P(\text{muerto}, M, 1) + P(\text{muerto}, M, 2) + P(\text{muerto}, M, 3) \\ &= 0,087 + 0,103 + 0,334 \\ &= 0,524 \end{aligned} \tag{2.10}$$

Para obtener  $P(M)$ , sumamos la supervivencia y la clase de cabina:

$$\begin{aligned} P(M) &= P(\text{muerto}, M, 1) + P(\text{muerto}, M, 2) + P(\text{muerto}, M, 3) \\ &\quad + P(\text{vivo}, M, 1) + P(\text{vivo}, M, 2) + P(\text{vivo}, M, 3) \\ &= 0,087 + 0,103 + 0,334 + 0,051 + 0,019 + 0,053 \\ &= 0,647 \end{aligned} \tag{2.11}$$

Para finalmente calcular  $P(\text{muerto}|M)$ :

$$P(\text{muerto}|M) = \frac{P(\text{muerto}, M)}{P(M)} = \frac{0,524}{0,647} = 0,810$$

Esto nos dice que el 81 por ciento de los pasajeros masculinos no sobrevivieron.

Un cálculo similar, que se muestra a continuación, nos dice la probabilidad de ser mujer y sobrevivir:

$$P(\text{vivo}|F) = \frac{P(\text{vivo}, F)}{P(F)} = \frac{0,263}{0,354} = 0,743$$

Vemos que las mujeres tenían muchas más probabilidades de sobrevivir que los hombres. He aquí un caso en el que la frase "las mujeres y los niños primero" fue realmente cierta. Te dejo como ejercicio calcular las probabilidades individuales en  $P(\text{vivo}|F)$ .

Hemos calculado  $P(\text{muerto}, M)$ , la probabilidad de ser un varón que no sobrevivió;  $P(M)$ , la probabilidad de ser hombre; y  $P(\text{muerto}|M)$ , la probabilidad de no sobrevivir siendo hombre. Hagamos un cálculo más basado en la Tabla 2-2. Encontremos  $P(\text{muerto o } M)$ , la probabilidad de no sobrevivir o ser hombre.

La ecuación 2.6 nos dice que esta es la probabilidad:

$$\begin{aligned} P(\text{muerto o } M) &= P(\text{muerto}) + P(M) - P(\text{muerto}, M) \\ &= 0,615 + 0,647 - 0,524 \\ &= 0,738 \end{aligned}$$

Si observamos la Ecuación 2.9 y la Ecuación 2.11, vemos que ambas tienen los mismos términos, los mismos términos resumidos en la Ecuación 2.10. Es por eso que debemos restar  $P(\text{muerto}, M)$  del cálculo de  $P(\text{muerto o } M)$  para evitar una doble contabilización.

Para resumir, entonces:

- La probabilidad conjunta es la probabilidad de que dos o más variables aleatorias tengan un conjunto específico de valores. La probabilidad conjunta suele representarse como una tabla.
- La probabilidad marginal de una variable aleatoria se encuentra sumando todos los valores posibles de las otras variables aleatorias.

La regla del producto con probabilidad condicional nos dice cómo calcular la probabilidad conjunta dada una probabilidad condicional y una probabilidad incondicional cuando tenemos dos variables aleatorias. Veamos ahora cómo utilizar la regla de la cadena de probabilidad para generalizar esa idea.

Regla de la cadena para la ecuación de probabilidad 2.8 nos dice cómo calcular la probabilidad conjunta de dos variables aleatorias en términos de la probabilidad condicional. Al utilizar la regla de la cadena para la probabilidad, podemos ampliar la ecuación 2.8 y calcular la probabilidad conjunta de más de dos variables aleatorias.

En su forma genérica, la regla de la cadena para la probabilidad conjunta de n variables aleatorias es la siguiente:

$$P(X_n, X_{n-1}, \dots, X_1) = \prod_{y_0=1}^{y_0-1} P(X_i \quad j=1 \quad x_j) \quad (2.12)$$

Aquí  $y$  se utiliza para indicar "y" para probabilidades conjuntas. La ecuación 2.12 parece impresionante, pero no es difícil de seguir, como veremos con algunos ejemplos. Necesito usar  $y$  en la ecuación para la parte conjunta de las probabilidades condicionales, pero en los ejemplos usaré una coma y verás el patrón lo suficientemente rápido.

Así es como la regla de la cadena divide una probabilidad conjunta con tres variables aleatorias:

$$\begin{aligned} P(X, Y, Z) &= P(X|Y, Z)P(Y, Z) \\ &= P(X|Y, Z)P(Y|Z)P(Z) \end{aligned}$$

La primera línea dice que la probabilidad de X, Y y Z es el producto de la probabilidad de X dados Y y Z y la probabilidad de Y y Z. Esta es la ecuación 2.8 con X para B e Y, Z para A. La segunda La línea aplica la regla de la cadena a P(Y, Z) para obtener P(Y|Z)P(Z). La regla se puede aplicar en secuencia, como una cadena, de ahí el nombre.

¿Qué pasa con una probabilidad conjunta con cuatro variables aleatorias? Obtenemos lo siguiente:

$$\begin{aligned} P(A, B, C, D) &= P(A|B, C, D)P(B, C, D) \\ &= P(A|B, C, D)P(B|C, D)P(C, D) \\ &= P(A|B, C, D)P(B|C, D)P(C|D)P(D) \end{aligned}$$

Analicemos un ejemplo que utiliza la regla de la cadena. Digamos que somos muy sociables y tenemos 50 personas en nuestra fiesta. Cuatro de las 50 personas estuvieron en Boston en el otoño. Escogemos a tres personas al azar. ¿Cuál es la probabilidad de que ninguno de ellos haya estado en Boston en el otoño?

Usaremos A<sub>i</sub> para indicar el evento de una persona que no ha estado en Boston. en el otoño. Por lo tanto, lo que queremos encontrar es P(A<sub>3</sub>, A<sub>2</sub>, A<sub>1</sub>), la probabilidad de que tres personas no hayan estado en Boston en el otoño. La regla de la cadena nos permite dividir esta probabilidad así:

$$P(A_3, A_2, A_1) = P(A_3|A_2, A_1)P(A_2|A_1)P(A_1)$$

Podemos trabajar intuitivamente con el lado derecho de esta ecuación. Mirar en P(A<sub>1</sub>). Esta es la probabilidad de elegir al azar a una persona en la sala que no haya estado en Boston en el otoño. Cuatro de las personas sí lo han hecho, por lo que 46 no lo han hecho, y vemos que P(A<sub>1</sub>) = 46/50. Una vez que hayamos elegido a una persona, necesitamos saber la probabilidad de seleccionar una segunda persona de las 49 restantes, eso es P(A<sub>2</sub>|A<sub>1</sub>) = 45/49. Sólo quedan 49 personas y no hemos seleccionado a ninguna de las cuatro que estuvieron en Boston en el otoño. Finalmente, seleccionamos a dos personas, por lo que hay 48 personas en la sala, 44 de las cuales no han estado en Boston en el otoño. Esto significa P(A<sub>3</sub>|A<sub>2</sub>, A<sub>1</sub>) = 44/48.

Ahora estamos listos para responder nuestra pregunta inicial. La probabilidad de seleccionar a tres personas de la sala sin que ninguna de ellas haya estado en Boston en el otoño es la siguiente:

$$\begin{aligned} P(A_3, A_2, A_1) &= P(A_3|A_2, A_1)P(A_2|A_1)P(A_1) \\ &= (44 \overline{48})(45 \overline{49})(46 \overline{50}) \\ &= 0,7745 \end{aligned}$$

Eso es un poco más del 77 por ciento.

Podemos comprobar si nuestro cálculo es razonable simulando muchos sorteos de tres personas. Este es el código que necesitamos:

---

```

nb = 0
norte = 100000
para i en el rango
    (N): s = np.random.randint(0,50,3)
    fail = False
    para t en rango(3):
        si (s[t] < 4): falla
            = Verdadero
        si (no falla): nb
            += 1
print("No hay Boston en otoño = %0.4f" % (nb/N,))

```

---

Realizaremos 100.000 simulaciones. Cada vez que seleccionemos a tres personas de 50 que no hayan estado en Boston en el otoño, incrementaremos nb. Simulamos la selección de tres personas eligiendo tres números enteros aleatorios en el rango [0, 50) y colocándolos en s. Luego, observamos cada uno de los tres números enteros y preguntamos si alguno es menor que cuatro. Si alguno lo es, decimos que seleccionamos a una persona que ha estado en Boston y configuramos falla en Verdadero. Si ninguno de los tres números enteros es menor que cuatro, esta simulación tuvo éxito. Cuando terminemos, imprimimos la fracción de simulaciones que muestran a tres personas que nunca fueron a Boston en el otoño.

Ejecutando este código producimos

---

Sin Boston en otoño = 0,7780

---

que está lo suficientemente cerca de nuestro valor calculado para darnos la confianza de que encontramos la respuesta correcta.

## Resumen

Este capítulo introdujo los fundamentos de la probabilidad. Exploramos conceptos básicos de probabilidad, incluidos espacios muestrales y variables aleatorias. Seguimos con algunos ejemplos de cuán pobres pueden ser los humanos en términos de probabilidad.

Después de eso, consideramos las reglas de probabilidad, con ejemplos. Las reglas nos llevaron a probabilidades conjuntas y marginales y finalmente a la regla de la cadena para probabilidades.

El siguiente capítulo continúa nuestro recorrido por la probabilidad, comenzando con las distribuciones de probabilidad y cómo tomar muestras de ellas y terminando con el teorema de Bayes, que nos muestra la forma correcta de comparar probabilidades condicionales.



# 3

## MÁS PROBABILIDAD



El capítulo 2 nos presentó los conceptos básicos de probabilidad. En este capítulo, continuar nuestra exploración de la probabilidad mediante centrarse en dos temas esenciales que a menudo se encuentran en el aprendizaje profundo y el aprendizaje automático: distribuciones de probabilidad y cómo tomar muestras de ellas, y Teorema de Bayes. El teorema de Bayes es uno de los conceptos más importantes de la teoría de la probabilidad y ha producido un cambio de paradigma en la forma en que muchos investigadores Piense en la probabilidad y cómo aplicarla.

### Distribuciones de probabilidad

Una distribución de probabilidad puede considerarse como una función que genera valores bajo demanda. Los valores generados son aleatorios: no sabemos cuáles. aparecerá uno, pero la probabilidad de que aparezca cualquier valor sigue un patrón general. forma. Por ejemplo, si lanzamos un dado estándar muchas veces y contamos cuántas veces que sale cada número, esperamos que a largo plazo, cada número

es igualmente probable. De hecho, ese es el objetivo de hacer el dado en la primera lugar. Por lo tanto, la distribución de probabilidad del dado se conoce como distribución uniforme, ya que cada número tiene la misma probabilidad de aparecer. Podemos imaginar otras distribuciones que favorecen un valor o rango de valores sobre otros, como un dado ponderado que podría aparecer como seis sospechosamente a menudo.

La razón principal del aprendizaje profundo para tomar muestras de una distribución de probabilidad es inicializar la red antes del entrenamiento. Las redes modernas seleccionan las ponderaciones iniciales y, a veces, los sesgos de diferentes distribuciones, la mayoría notablemente uniforme y normal. La distribución uniforme nos resulta familiar y Hablaré de la distribución normal, una distribución continua, más adelante.

En esta sección presentaré varios tipos diferentes de distribuciones de probabilidad. Nuestro objetivo es comprender la forma de la distribución y aprender cómo extraer muestras usando NumPy. Comenzaré con histogramas para mostrar usted que a menudo podemos tratar los histogramas como aproximaciones de una probabilidad distribución. Luego discutiré las distribuciones de probabilidad discretas comunes. Estos son distribuciones que devuelven valores enteros, como 3 o 7. Por último, cambiaré a distribuciones continuas que producen números de punto flotante, como 3.8 o 7.592.

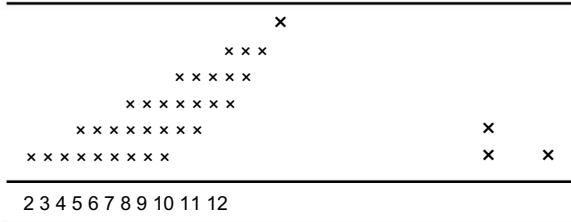
## Histogramas y probabilidades

Eche un vistazo a la Tabla 3-1, que vimos en el Capítulo 2.

Tabla 3-1: Número de combinaciones de dos dados que conducen a Diferentes sumas (copiadas de la tabla 2-1)

Combinaciones de suma		contar probabilidad	
2	1 +	1	0.0278
3	1 1 + 2, 2 +	2	0.0556
4	1 1 + 3, 2 + 2, 3 + 1 3 1 + 4, 2 + 3, 3 + 2, 4 + 1 4	6	0.0833
5	1 + 5, 2 + 4, 3 + 3, 4 + 2, 5 + 1 5 1 + 6, 2 + 5, 3 +	10	0.1111
6	4, 4 + 3, 5 + 2, 6 + 1 6 2 + 6, 3 + 5, 4 + 4, 5 + 3,	15	0.1389
7	6 + 2 5 3 + 6, 4 + 5, 5 + 4, 6 + 3 4 + 6, 5 + 5, 6 +	21	0.1667
8	4 5 + 6, 6 + 5 6 + 6	15	0.1389
9		4	0.1111
10		3	0.0833
11		2	0.0556
12		1	0.0278
		1.36	1.0000

Muestra cómo dos dados suman diferentes sumas. No mires lo real valores; Fíjate en la forma que forman las posibles combinaciones. Si cortamos el últimas dos columnas, gire la tabla hacia la izquierda y reemplace cada suma con una "X", deberíamos ver algo como lo siguiente.



Puedes ver que hay una forma definida y simetría en el número. de formas de llegar a cada suma. Este tipo de gráfico se llama histograma. Un histograma es un gráfico que cuenta la cantidad de cosas que se encuentran en contenedores discretos. Para En la Tabla 3-1, los contenedores son los números del 2 al 12. El recuento es una forma posible para conseguir esa suma. Los histogramas a menudo se representan como gráficos de barras, generalmente barras verticales, aunque no es necesario que lo sean. La tabla 3-1 es básicamente un histograma horizontal. La cantidad de contenedores que se utilizan en el histograma depende del creador. Si usted Si utiliza muy pocos, el histograma aparecerá en bloques y es posible que no revele los detalles necesarios porque todas las características interesantes se han agrupado en el mismo contenedor. Usar demasiados contenedores y el histograma será escaso, y muchos contenedores no tendrán cuentas.

Generemos algunos histogramas. Primero, tomaremos muestras aleatorias de números enteros. en [0,9] y contar cuántos de cada número entero obtenemos. El código para esto es directo:

---

```
>>> importar números como np
>>> n = np.aleatorio.randint(0,10,10000)
>>> h = np.bincount(n)
>>> h

matriz([ 975, 987, 987, 1017, 981, 1043, 1031, 988, 1007, 984])
```

---

Primero establecemos n en una matriz de 10.000 números enteros en [0, 9]. Luego usamos np .bincount para contar cuántos de cada dígito tenemos. Vemos que esta carrera nos dio 975 ceros y 984 nueves. Si el generador pseudoaleatorio NumPy está haciendo su trabajo, esperamos, en promedio, tener 1 000 de cada dígito en una muestra de 10 000 dígitos. Esperamos alguna variación, pero la mayoría de los valores están cerca. suficientes para 1.000 para ser convincente.

Los conteos anteriores nos dicen cuántas veces apareció cada dígito. Si dividimos cada contenedor de un histograma por el total de todos los contenedores, pasamos de recuentos simples a la probabilidad de que aparezca ese contenedor. Para los dígitos aleatorios arriba, obtenemos las probabilidades con

---

```
>>> h = h / h.suma()
>>> h

matriz([0,0975, 0,0987, 0,0987, 0,1017, 0,0981, 0,1043, 0,1031, 0,0988,
       0,1007, 0,0984])
```

---

lo que nos dice que cada dígito apareció con una probabilidad de aproximadamente 0,1, o 1 de 10. Este truco de dividir los valores del histograma por la suma de los recuentos en el histograma nos permite estimar distribuciones de probabilidad a partir de muestras. También nos indica la probabilidad de que aparezcan valores particulares al tomar muestras de cualquier proceso que generó los datos utilizados para hacer el histograma.

Debes tener en cuenta que dije que podíamos estimar la distribución de probabilidad a partir de un conjunto de muestras extraídas de él. Cuanto mayor sea el número de muestras, mayor más cerca estará la distribución de probabilidad estimada de la población real distribución que genera las muestras. Nunca llegaremos a la distribución real de la población, pero dado el límite de un número infinito de muestras, podemos acercarnos tanto como necesitemos.

Los histogramas se utilizan con frecuencia para observar la distribución de valores de píxeles en una imagen. Hagamos un trazado del histograma de los píxeles en dos imágenes. Puede encontrar el código en el archivo ricky.py. (No lo mostraré aquí, ya que no agregar a la discusión). Las imágenes utilizadas son dos imágenes de ejemplo en escala de grises, incluido con SciPy en scipy.misc. La primera muestra a personas subiendo escaleras (ascenso), y el segundo es la cara de un mapache joven (cara), como se muestra en Figura 3-1.

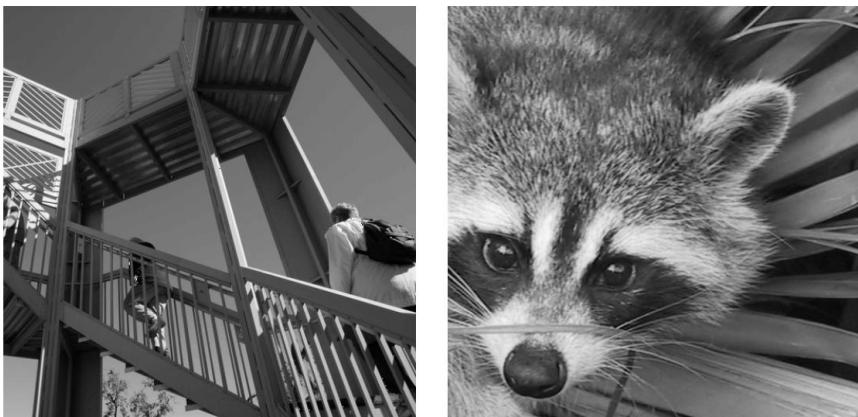


Figura 3-1: Personas ascendiendo (izquierda) y mapache "Ricky" (derecha)

La Figura 3-2 proporciona un gráfico de los histogramas de cada imagen, como probabilidades. Muestra dos distribuciones muy diferentes de valores de nivel de gris en las imágenes. Para la cara del mapache, la distribución es más extendida y plana, mientras que la imagen de ascenso tiene un pico alrededor del nivel de gris 128 y algunos píxeles brillantes. Las distribuciones nos dicen que si elegimos un píxel aleatorio en la imagen de la cara, lo más probable es que obtengamos uno alrededor del nivel de gris 100, pero un píxel arbitrario en la imagen ascendente estará, con una alta probabilidad relativa, más cerca del nivel de gris 100. nivel de gris 128.

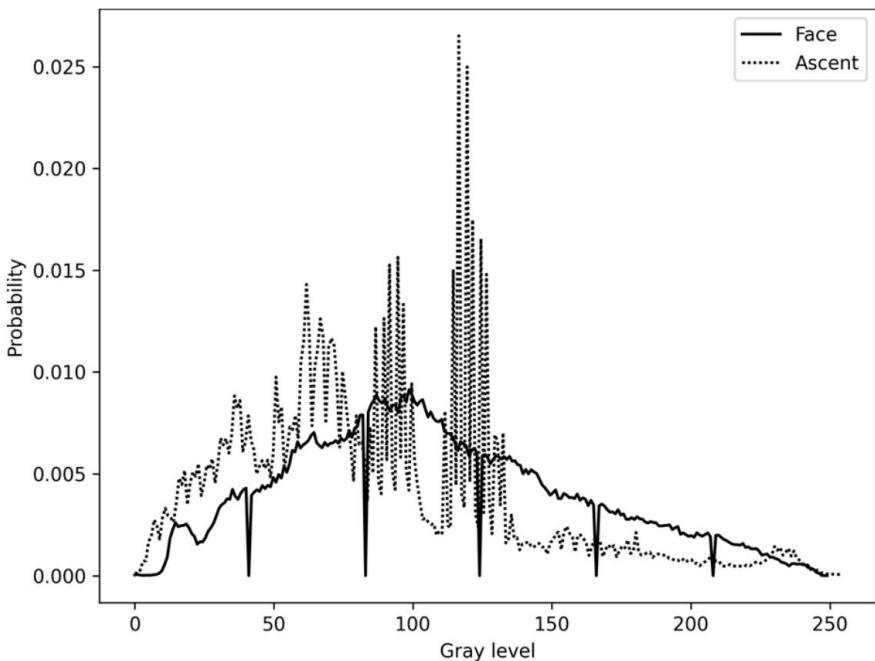


Figura 3-2: Histogramas como probabilidades para dos imágenes de muestra en escala de grises de  $512 \times 512$  píxeles

Nuevamente, los histogramas cuentan cuántos elementos caen en los contenedores predefinidos. Vimos en las imágenes que el histograma como distribución de probabilidad nos dice qué probabilidad tenemos de obtener un valor de nivel de gris particular si seleccionamos un píxel aleatorio. Asimismo, la distribución de probabilidad para los dígitos aleatorios en El ejemplo anterior nos dice la probabilidad de obtener cada dígito cuando solicite un número entero aleatorio en el rango  $[0, 9]$ .

Los histogramas son representaciones discretas de una distribución de probabilidad. Echemos un vistazo ahora a las distribuciones discretas más comunes.

### Distribuciones de probabilidad discreta

Ya nos hemos encontrado varias veces con la distribución discreta más común: es la distribución uniforme. Ese es el que obtenemos naturalmente tirar dados o lanzar monedas. En la distribución uniforme, todos los resultados posibles son igualmente probables. Un histograma de una simulación de un dibujo de proceso de una distribución uniforme es plana; todos los resultados aparecen con más o menos la misma frecuencia. Veremos la distribución uniforme nuevamente cuando miremos en distribuciones continuas. Por ahora, piensa en los dados.

Veamos algunas otras distribuciones discretas.

### La distribución binomial

Quizás la segunda distribución discreta más común sea la distribución binomial. Esta distribución representa el número esperado de eventos que suceden en un número dado de ensayos si cada evento tiene una probabilidad específica.

Matemáticamente, la probabilidad de que ocurran  $k$  eventos en  $n$  ensayos si la probabilidad de que ocurra el evento es  $p$  se puede escribir como

$$P(X = k) = (n) p^k (1 - p)^{n-k}$$

Por ejemplo, ¿cuál es la probabilidad de obtener tres caras seguidas al lanzar una moneda justa tres veces? A partir de la regla del producto, sabemos que la probabilidad es

$$P(HHH) = (1/2)(1/2)(1/2) = \frac{1}{8} = 0,125$$

Usando la fórmula binomial, obtenemos la misma respuesta calculando

$$P(HHH) = (3) (0,5)^3 (1 - 0,5)^{3-3} = 0,125$$

Hasta ahora, no es particularmente útil. Sin embargo, ¿qué pasa si la probabilidad del evento no es 0,5? ¿Qué pasa si tenemos un evento, digamos la probabilidad de que una persona gane? Hagamos un trato al no cambiar de puerta, y queremos saber la probabilidad de que 7 personas de 13 ganen al no cambiar su suposición? Sabemos que la probabilidad de ganar el juego sin cambiar de puerta es 1/3; eso es  $p$ .

Entonces tenemos 13 ensayos ( $n$ ) y 7 ganadores ( $k$ ). La fórmula binomial nos dice que la probabilidad es

$$P(X = 7) = (13) \left(\frac{1}{3}\right)^7 \left(1 - \frac{1}{3}\right)^{13-7} = 0,0689$$

y, si los jugadores cambian de puerta,

$$P(X = 7) = (13) \left(\frac{2}{3}\right)^7 \left(1 - \frac{2}{3}\right)^{13-7} = 0,1378$$

La fórmula binomial nos da la probabilidad de un número determinado de eventos en un número determinado de ensayos para una probabilidad específica por evento. Si fijamos  $n$  y  $p$  y variamos  $k$ ,  $0 \leq k \leq n$ , obtenemos la probabilidad para cada valor de  $k$ .

Esto nos da la distribución. Por ejemplo, sean  $n = 5$  y  $p = 0,3$ , entonces  $0 \leq k \leq 5$  con la probabilidad para cada valor de  $k$  como

$$P(X = 0) = (5)(0,3)^0(1 - 0,3)^{5-0} = 0,1681$$

$$P(X = 1) = (5)(0,3)^1(1 - 0,3)^{5-1} = 0,3601$$

$$P(X = 2) = (5)(0,3)^2(1 - 0,3)^{5-2} = 0,3087$$

$$P(X = 3) = (5)(0,3)^3(1 - 0,3)^{5-3} = 0,1323$$

$$P(X = 4) = (5)(0,3)^4(1 - 0,3)^{5-4} = 0,0283$$

$$P(X = 5) = (5)(0,3)^5(1 - 0,3)^{5-5} = 0,0024$$

Teniendo en cuenta el redondeo, esto suma 1,0, como sabemos que debe ser porque la suma de probabilidades en un espacio muestral completo es siempre 1,0. Observe que calculamos todos los valores posibles para la distribución binomial cuando  $n = 5$ . En conjunto, esto especifica la función de masa de probabilidad (pmf). La función de masa de probabilidad nos dice la probabilidad asociada con todos los resultados posibles.

La distribución binomial está parametrizada por  $n$  y  $p$ . Para  $n = 5$  y  $p = 0,3$ , vemos en los resultados anteriores que una muestra aleatoria de dicha distribución binomial arrojará 1 con mayor frecuencia (aproximadamente el 36 por ciento de las veces).

¿Cómo podemos extraer muestras de una distribución binomial? En NumPy, sólo necesitamos llamar a la función binomial en el módulo aleatorio :

---

```
>>> t = np.random.binomial(5, 0.3, tamaño=1000)
>>> s = np.bincount(t)
>>> s

matriz([159, 368, 299, 155, 17, 2]) >>> s /
s.sum()

matriz([0.159, 0.368, 0.299, 0.155, 0.017, 0.002])
```

---

Pasamos binomial el número de intentos (5) y la probabilidad de éxito. para cada ensayo (0,3). Luego solicitamos 1000 muestras de una distribución binomial con estos parámetros. Al usar np.bincount, vemos que el valor devuelto más comúnmente fue 1, como calculamos anteriormente. Al utilizar nuestro truco de suma de histogramas, obtenemos una probabilidad de 0,368 de seleccionar un 1, cercana al 0,3601 que calculamos.

### La distribución de Bernoulli

La distribución de Bernoulli es un caso especial de la distribución binomial. En este caso, fijamos  $n = 1$ , lo que significa que solo hay una prueba. Los únicos valores que podemos muestrear son 0 o 1; O el evento ocurre o no. Por ejemplo, con  $p = 0,5$ , obtenemos

---

```
>>> t = np.random.binomial(1, 0.5, tamaño=1000) >>>
np.bincount(t)
matriz([496, 504])
```

---

Esto es razonable, ya que una probabilidad de 0,5 significa que estamos lanzando una moneda justa y vemos que la proporción de cara o cruz es aproximadamente igual.

Si cambiamos a  $p = 0,3$ , obtenemos

---

```
>>> t = np.random.binomial(1, 0.3, tamaño=1000) >>>
np.bincount(t)
matriz([665, 335])
>>> 335/1000
0.335
```

---

De nuevo, cerca de 0,3, como esperamos ver.

Utilice muestras de una distribución binomial cuando desee simular eventos con una probabilidad conocida. Con la forma de Bernoulli, podemos muestrear resultados binarios, 0 o 1, donde la probabilidad del evento no tiene por qué ser la de lanzar una moneda justa, 0,5.

### La distribución de Poisson

A veces, no conocemos la probabilidad de que ocurra un evento para una prueba en particular. En lugar de ello, podríamos conocer el número promedio de eventos que ocurren en algún intervalo, digamos de tiempo. Si el número promedio de eventos que ocurren durante un tiempo es ( $\lambda$ ), entonces la probabilidad de que ocurran  $k$  eventos en ese intervalo es

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Esta es la distribución de Poisson y es útil para modelar eventos como la desintegración radiactiva o la incidencia de fotones en un detector de rayos X durante un período de tiempo. Para muestrear eventos de acuerdo con esta distribución, utilizamos poisson del módulo aleatorio . Por ejemplo, supongamos que durante un intervalo de tiempo hay cinco eventos en promedio (= 5). ¿Qué tipo de distribución de probabilidad obtenemos usando la distribución de Poisson? En código,

---

```
>>> t = np.random.poisson(5, tamaño=1000)
>>> s = np.bincount(t)
>>> s
matriz([ 6, 36, 83, 135, 179, 173, 156, 107, 58, 40, 20, 4,
          0,      0,      1])
```

2,

---

```
>>> t.max()
15
>>> s = s / s.sum()
>>> s
matriz([0,006, 0,036, 0,083, 0,135, 0,179, 0,173, 0,156, 0,107, 0,058,
       0,04   , 0,02   , 0,004, 0,002, 0.           , 0.     , 0,001])
```

---

Aquí vemos que, a diferencia de la distribución binomial, que no podía seleccionar más de n eventos, la distribución de Poisson puede seleccionar números de eventos que superan el valor de . En este caso, el mayor número de eventos en el El intervalo de tiempo fue de 15, que es tres veces el promedio. Encontrarás que el El número más frecuente de eventos está alrededor del promedio de cinco, como podría esperarse, pero son posibles desviaciones significativas del promedio.

El rodillo de dados de carga rápida

¿Qué pasa si necesitamos extraer muestras de acuerdo con una distribución discreta arbitraria? Anteriormente vimos algunos histogramas basados en imágenes. En ese caso, nosotros Podría tomar muestras de la distribución representada por el histograma seleccionando píxeles de la imagen al azar. Pero ¿y si quisieramos muestrear números enteros? ¿Según pesos arbitrarios? Para ello podemos utilizar el nuevo Fast Loaded Rodillo de dados de Saad, et al.<sup>1</sup>

El rodillo de dados de carga rápida (FLDR) nos permite especificar un valor discreto arbitrario. distribución y luego extraer muestras de ella. El código está en Python y de forma libre. disponible. (Consulte <https://github.com/probcomp/fast-loaded-dice-roller/>.) Le mostraré cómo utilizar el código para muestrear según una distribución genérica. Recomiendo descargar solo los archivos fldr.py y fldrf.py del repositorio de GitHub en lugar de ejecutar setup.py. Además, edite las líneas de importación .fldr en fldrf.py para eliminar el "." entonces ellos lean

---

```
desde fldr importar fldr_preprocess_int
desde fldr importar fldr_s
```

---

El uso de FLDR requiere dos pasos. La primera es indicarle la distribución particular de la que desea realizar la muestra. La distribución se define como ratios. (Para nuestros propósitos, usaremos probabilidades reales, lo que significa que nuestra distribución siempre sumará 1.0.) Este es el paso de preprocesamiento, que solo necesitamos hacer una vez por cada distribución. Después de eso, podemos tomar muestras. Un ejemplo será aclarar:

---

```
>>> desde fldr importar fldr_preprocess_float_c
>>> desde fldr importar fldr_sample
>>> x = fldr_preprocess_float_c([0.6,0.2,0.1,0.1])
>>> t = [fldr_sample(x) para i en el rango(1000)]
```

---

1. Feras A. Saad, Cameron E. Freer, Martin C. Rinard y Vikash K. Mansinghka, "The Fast Rodillo de dados cargado: un muestreador exacto casi óptimo para distribuciones de probabilidad discretas", en AISTATS 2020: Actas de la 23<sup>a</sup> Conferencia Internacional sobre Inteligencia Artificial y Estadísticas, Actas de Machine Learning Research 108, Palermo, Sicilia, Italia, 2020.

---

```
>>> np.bincount(t)
matriz([598, 190, 108, 104])
```

---

Primero, importamos las dos funciones FLDR que necesitamos: fldr\_preprocess\_float\_c y fldr\_sample. Luego definimos la distribución usando una lista de cuatro números. Cuatro números implican que las muestras serán números enteros en [0, 3]. Sin embargo, a diferencia de una distribución uniforme, aquí especificamos que queremos cero el 60 por ciento de las veces, uno el 20 por ciento de las veces y dos y tres el 10 por ciento de las veces cada uno. La información que FLDR necesita tomar de la distribución se devuelve en x.

Llamar a fldr\_sample devuelve una única muestra de la distribución. Observe dos cosas: primero, necesitamos pasar x y segundo, FLDR no usa NumPy, por lo que para extraer 1000 muestras, usamos una lista de comprensión estándar de Python. Las 1.000 muestras están en la lista, t. Finalmente, generamos el histograma y vemos que casi el 60 por ciento de las muestras son cero y un poco más del 10 por ciento son tres, como pretendíamos.

Usemos el histograma de la imagen de la cara del mapache que usamos anteriormente para ver si FLDR seguirá una distribución más compleja. Cargaremos la imagen, generaremos el histograma, lo convertiremos en una distribución de probabilidad y usaremos las probabilidades para configurar FLDR. Después de eso, extraeremos 25 000 muestras de la distribución, calcularemos el histograma de las muestras y trazaremos ese histograma junto con el histograma original para ver si FLDR sigue la distribución real que le damos. El código que necesitamos es

---

```
de scipy.misc importar cara
im = cara(True)
b = np.bincount(im.ravel(), minlength=256)
b = b / b.sum()
x = fldr_preprocess_float_c(list(b)) t
= [fldr_sample (x) para i en el rango(25000)]
q = np.bincount(t, minlength=256)
q = q / q.sum()
```

---

Al ejecutar este código nos queda b, una distribución de probabilidad de la histograma de la imagen de la cara, y q, la distribución creada a partir de 25.000 muestras de la distribución FLDR. La Figura 3-3 nos muestra un gráfico de las dos distribuciones.

La línea continua en la Figura 3-3 es la distribución de probabilidad que proporcionamos a fldr\_preprocess\_float\_c representa la distribución de niveles de gris (intensidades) en la imagen del mapache. La línea discontinua es el histograma de las 25.000 muestras de esta distribución. Como podemos ver, siguen la distribución solicitada con el tipo de variación que esperamos de un número tan pequeño de muestras. Como ejercicio, cambie el número de muestras de 25 000 a 500 000 y trace las dos curvas. Verás que ahora están prácticamente uno encima del otro.

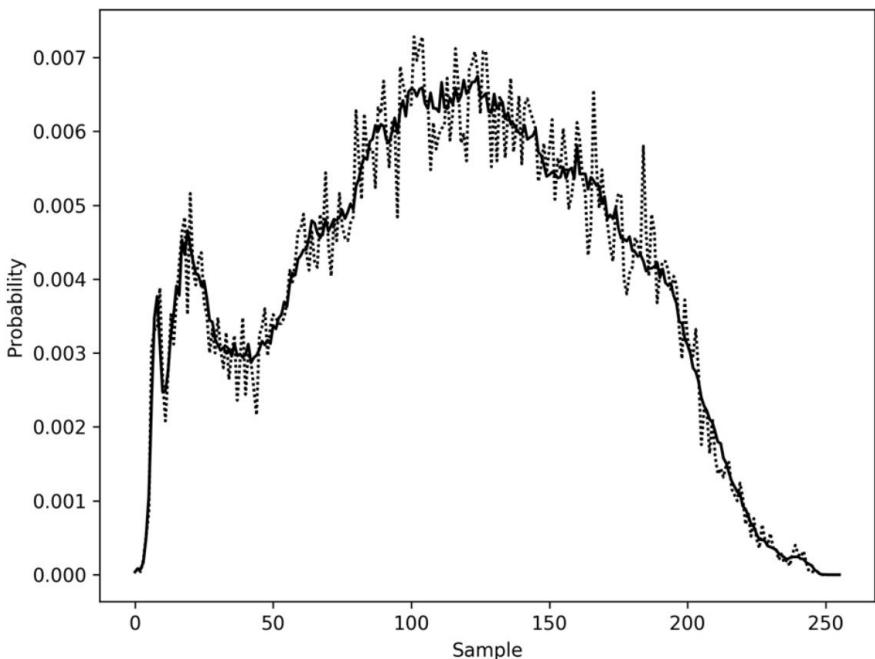


Figura 3-3: Comparación de la distribución del Fast Loaded Dice Roller (discontinua) con la distribución generada a partir de la imagen facial de SciPy (sólida)

Las distribuciones discretas generan números enteros con probabilidades específicas. Dejémoslos ahora y consideremos distribuciones de probabilidad continuas, que en su lugar devuelven valores de punto flotante.

#### Distribuciones de probabilidad continua

Aún no he analizado las probabilidades continuas en este capítulo. En parte, no hacerlo fue para que los conceptos detrás de la probabilidad fueran más fáciles de seguir. Una distribución de probabilidad continua, como una discreta, tiene una forma particular. Sin embargo, en lugar de asignar una probabilidad a un valor entero específico, como vimos anteriormente, la probabilidad de seleccionar un valor particular de una distribución continua es cero. La probabilidad de un valor específico, un número real, es cero porque hay un número infinito de valores posibles de una distribución continua; esto significa que no se puede seleccionar ningún valor en particular. Más bien, hablamos de la probabilidad de seleccionar valores en un rango específico de valores.

Por ejemplo, la distribución continua más común es la uniforme distribución sobre  $[0, 1]$ . Esta distribución devuelve cualquier número real en ese rango. Aunque la probabilidad de devolver un número real específico es cero, podemos hablar de la probabilidad de devolver un valor en un rango, como  $[0, 0,25]$ .

Consideremos nuevamente la distribución uniforme sobre  $[0, 1]$ . Sabemos que la suma de todas las probabilidades individuales de cero a uno es 1,0. Entonces, ¿cuál es la probabilidad de muestrear un valor de esta distribución y que ese valor esté en el rango  $[0, 0,25]$ ? Todos los valores son igualmente probables y todos suman 1,0, por lo que debemos tener un 25 por ciento de posibilidades de devolver un valor en  $[0, 0,25]$ . De manera similar, tenemos un 25 por ciento de posibilidades de devolver un valor en  $[0,75, 1]$ , ya que eso también cubre  $1/4$  del rango posible.

Cuando hablamos de sumar cosas infinitamente pequeñas en un rango, estamos hablando de integración, la parte del cálculo que no cubriremos en este libro.

Conceptualmente, sin embargo, podemos entender lo que sucede si pensamos en una distribución discreta en el límite donde el número de valores que puede devolver llega al infinito, y sumamos las probabilidades en algún rango.

También podemos pensar en esto gráficamente. La figura 3-4 muestra las distribuciones de probabilidad continua que analizaremos.

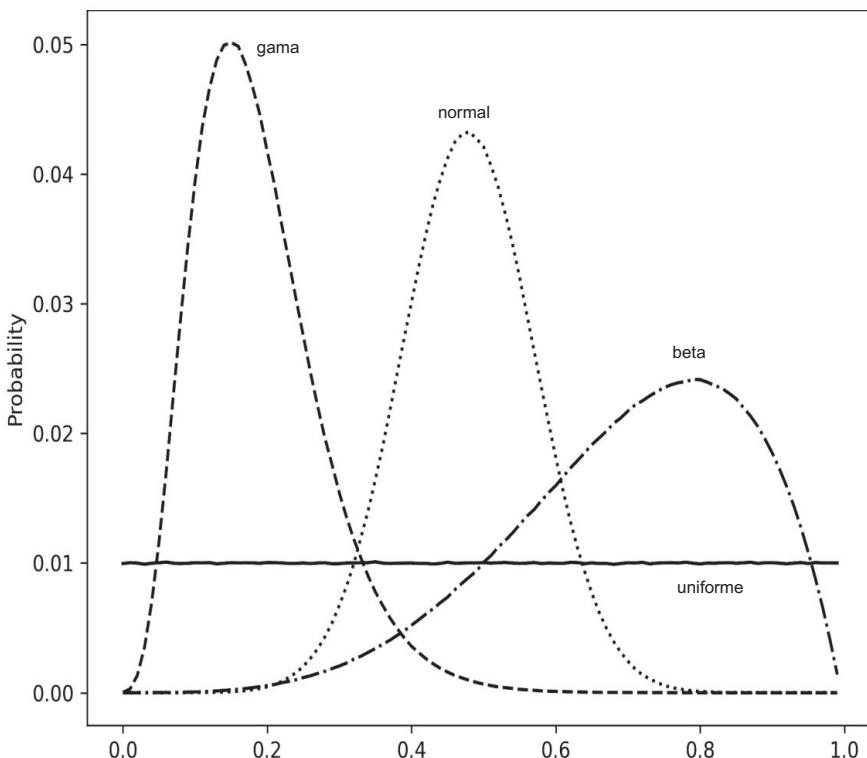


Figura 3-4: Algunas distribuciones de probabilidad continua comunes

Para obtener la probabilidad de muestrear un valor en algún rango, sumamos el área bajo la curva en ese rango. De hecho, esto es precisamente lo que hace la integración; el símbolo de integración ( $\int$ ) no es más que una elegante "S" para suma. Es la versión continua de  $\Sigma$  para sumar valores discretos.

Las distribuciones de la Figura 3-4 son las más comunes que encontrará, aunque hay muchas otras lo suficientemente útiles como para darles nombres.

Todas estas distribuciones tienen funciones de densidad de probabilidad (dpd) asociadas, funciones de forma cerrada que generan las probabilidades que dará el muestreo de la distribución. En lugar de eso, generé las curvas en la Figura 3-4 usando el código en el archivo continuo.py. Las curvas son estimaciones de las funciones de densidad de probabilidad y las creé a partir del histograma de una gran cantidad de muestras. Lo hice intencionalmente para demostrar que las funciones aleatorias de NumPy que toman muestras de estas distribuciones hacen lo que afirman.

Preste poca atención al eje x en la Figura 3-4. Las distribuciones tienen diferentes rangos de producción; están escalados aquí para que quiepan todos en el gráfico.

Lo importante a destacar son sus formas. La distribución uniforme es, bueno, uniforme en todo el rango. La curva normal, también llamada frecuentemente gaussiana o curva de campana, es la segunda distribución más común utilizada en el aprendizaje profundo. Por ejemplo, la estrategia de inicialización de He para redes neuronales toma muestras de pesos iniciales de una distribución normal.

Vale la pena considerar el código que genera los datos de la Figura 3-4, ya que nos muestra cómo usar NumPy para obtener muestras:

---

```
norte = 10000000
B = 100

t = np.random.random(N) u =
np.histogram(t, bins=B)[0] u = u / u.sum()
= np.random.normal(0,
1, tamaño=N) n = np.histogram(t, bins=B)[0] n =
n / n.sum() t = np.random.gamma(5.0,
tamaño=N) g =
np.histogram(t, bins=B) [0] g = g / g.sum() t =
np.random.beta(5.2, tamaño=N) b =
np.histogram(t, bins=B)
[0] b = b / b.sum ()
```

---

### NOTA

Estamos usando las funciones clásicas de NumPy aquí, no las funciones más nuevas basadas en Generador . NumPy actualizó el código numérico pseudoaleatorio en versiones recientes, pero la sobrecarga de usar el nuevo código restará valor a lo que queremos ver aquí. A menos que se tome muy en serio la generación de números pseudoaleatorios, las funciones más antiguas y el generador de números pseudoaleatorios Mersenne Twister en el que se basan serán más que adecuados.

Para hacer los gráficos, primero utilizamos 10 millones de muestras de cada distribución (N). Luego, usamos 100 contenedores en el histograma (B). Nuevamente, el rango del eje x al trazar no es de interés aquí, solo las formas de las curvas.

Las muestras uniformes utilizan el azar, una función que hemos visto antes. Pasar las muestras al histograma y aplicar el truco de “dividir por la suma” crea los datos de la curva de probabilidad (u). También repetimos este proceso para las distribuciones Gaussiana (normal), Gamma (gamma) y Beta (beta) .

Notarás que normal, gamma y beta aceptan argumentos. Estas distribuciones están parametrizadas; su forma se altera al cambiar estos parámetros. Para la curva normal, el primer parámetro es la media ( ) y el segundo es la desviación estándar ( ). Alrededor del 68 por ciento de la curva normal se encuentra dentro de una desviación estándar de la media, [ – + ]. La curva normal es omnipresente en matemáticas y en la naturaleza, y se podría escribir un libro entero sólo sobre ella. Siempre es simétrico alrededor de su valor medio. La desviación estándar controla qué tan ancha o estrecha es la curva.

La distribución gamma también está parametrizada. Acepta dos parámetros: la forma (k) y la escala ( ). Aquí, k = 5, y la escala se deja en su valor predeterminado de = 1. A medida que aumenta la forma, la distribución gamma se parece cada vez más a una gaussiana, con un relieve que se mueve hacia el centro de la distribución. El parámetro de escala afecta el tamaño horizontal del bullo.

Asimismo, la distribución beta utiliza dos parámetros, a y b. Aquí, a = 5 y b = 2. Si a > b, la joroba de la distribución está a la derecha; si está al revés, está a la izquierda. Si a = b, la distribución beta se convierte en la distribución uniforme.

La flexibilidad de la distribución beta la hace bastante útil para simular diferentes procesos, siempre que pueda encontrar valores de a y b que se aproximen a la distribución de probabilidad que desea. Sin embargo, dependiendo de la precisión que requiera, el nuevo rodillo de dados de carga rápida que vimos en la sección anterior podría ser una mejor opción en la práctica si tiene una aproximación de distribución discreta suficientemente detallada de la distribución continua.

La tabla 3-2 nos muestra las funciones de densidad de probabilidad para las distribuciones normal, gamma y beta. Un ejercicio para el lector es utilizar estas funciones para recrear la Figura 3-4. Sus resultados serán aún más suaves que las curvas de la figura. Puede calcular la integral B(a, b) en la Tabla 3-2 utilizando la función `scipy.special.beta`. Para  $\Gamma(k)$ , consulte `scipy.special.gamma`. Además, si el argumento de la función  $\Gamma$  es un número entero,  $\Gamma(n+1) = n!$ , entonces  $\Gamma(5) = \Gamma(4+1) = 4! = 24$ .

Tabla 3-2: Funciones de densidad de probabilidad para las distribuciones Normal, Gamma y Beta

normal	$p(x) = \sqrt{2} \frac{1}{2\pi} e^{-\frac{(x-\mu)^2}{2}}$
gama	$p(x) = x^{k-1} \frac{e^{-\frac{x}{\theta}}}{k\Gamma(k)}, \Gamma(k) = \int_0^{\infty} t^{k-1} e^{-t} dt$
beta	$p(x) = \frac{1}{B(a,b)} x^{a-1} (1-x)^{b-1}, B(a,b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt$

Si está interesado en formas de muestrear valores de estas distribuciones, mi libro *Random Numbers and Computers* (Springer, 2018) analiza estas distribuciones y otras con más profundidad de la que podemos proporcionar aquí, incluidas implementaciones en C para generar muestras a partir de ellas. . Por ahora, examinemos uno de los teoremas más importantes de la teoría de la probabilidad.

### Teorema del límite central

Imaginemos que extraemos  $N$  muestras de alguna distribución y calculamos el valor medio,  $m$ . Si repetimos este ejercicio muchas veces, obtendremos un conjunto de valores medios,  $\{m_0, m_1, \dots\}$ , cada uno de un conjunto de muestras de la distribución. No importa si  $N$  es el mismo cada vez, pero  $N$  no debería ser demasiado pequeño. La regla general es que  $N$  debe tener al menos 30 muestras.

El teorema del límite central establece que el histograma o distribución de probabilidad generada a partir de este conjunto de medias muestrales, las  $m$ , se aproximará a una forma gaussiana independientemente de la forma de la distribución de la que se extrajeron las muestras en primer lugar.

Por ejemplo, este código

---

```
M = 10000
m = np.zeros(M)
para i en el rango(M):
    t = np.random.beta(5,2,size=M) m[i]
    = t.mean()
```

---

Crea 10.000 conjuntos de muestras a partir de una distribución beta, Beta(5,2), cada uno con 10.000 muestras. La media de cada conjunto de muestras se almacena en  $m$ . Si ejecutamos este código y trazamos el histograma de  $m$ , obtenemos la Figura 3-5.

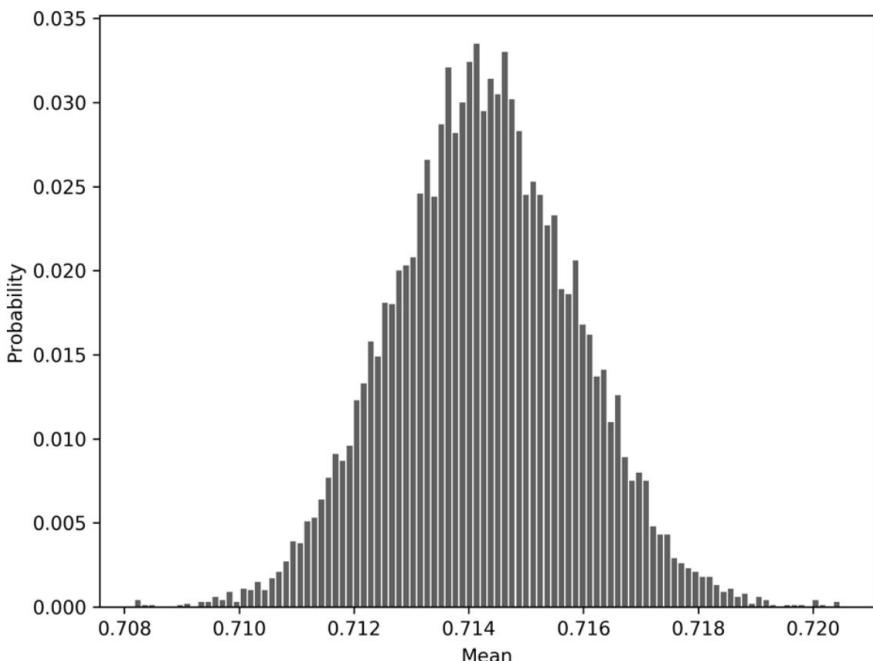


Figura 3-5: La distribución de los valores medios de 10.000 conjuntos de muestras de 10.000 de Beta(5,2)

La forma de la figura 3-5 es decididamente gaussiana. Nuevamente, la forma es una secuencia del teorema del límite central y no depende de la forma

de la distribución subyacente. La Figura 3-5 nos dice que las medias muestrales de muchos conjuntos de muestras de Beta(5,2) tienen una media de aproximadamente 0,714. La media de las medias de muestra (`m.mean()`) es 0,7142929 para una ejecución del código anterior.

Existe una fórmula para calcular el valor medio de una distribución Beta.

Se sabe que el valor medio poblacional de una distribución Beta(5,2) es  $a/(a + b) = 5/(5 + 2) = 5/7 = 0,714285$ . La media del gráfico de la figura 3-5 es una medida de la media poblacional verdadera, de la cual las muchas medias de las muestras Beta(5,2) son sólo estimaciones.

Expliquemos esto nuevamente para seguir realmente lo que está pasando. Para cualquier distribución, como la distribución Beta (5,2), si extraemos N muestras, podemos calcular la media de esas muestras, un solo número. Si repetimos este proceso para muchos conjuntos de N muestras, cada una con su propia media, y hacemos un histograma de la distribución de las medias que medimos, obtendremos un gráfico como el de la Figura 3-5.

Ese gráfico nos dice que todas las medias muestrales están agrupadas en torno a un valor medio. El valor medio de las medias es una medida de la media poblacional. Es la media que obtendríamos si pudiéramos extraer un número infinito de muestras de la distribución. Si cambiamos el código anterior para usar la distribución uniforme, obtendremos una media poblacional de 0,5. De manera similar, si cambiamos a una distribución gaussiana con una media de 11, el histograma resultante estará centrado en 11.

Probemos esta afirmación nuevamente pero esta vez con una distribución discreta.

Usemos el Fast Loaded Dice Roller para generar muestras de una distribución discreta asimétrica usando este código:

---

```
de fldr importar fldr_preprocess_float_c
de fldr importar fldr_sample
z = fldr_preprocess_float_c([0.1,0.6,0.1,0.1,0.1]) m
= np.zeros(M)
para i en rango(M):
    t = np.array([fldr_sample(z ) para i en el rango(M)])
    m[i] = t.mean()
```

---

La Figura 3-6 muestra la distribución discreta (arriba) y la distribución correspondiente de las medias muestrales (abajo).

A partir de la función de masa de probabilidad, podemos ver que el valor más frecuente que esperamos de la muestra es 1, con una probabilidad del 60 por ciento. Sin embargo, la cola de la derecha significa que también obtendremos los valores del 2 al 4 aproximadamente el 30 por ciento de las veces. La media ponderada de estos es  $0,6(1) + 0,1(2) + 0,1(3) + 0,1(4) = 1,5$ , que es precisamente la media de la distribución de la muestra en la parte inferior de la Figura 3-6. El teorema del límite central funciona. Volveremos a examinar el teorema del límite central en el capítulo 4 cuando analicemos la prueba de hipótesis.

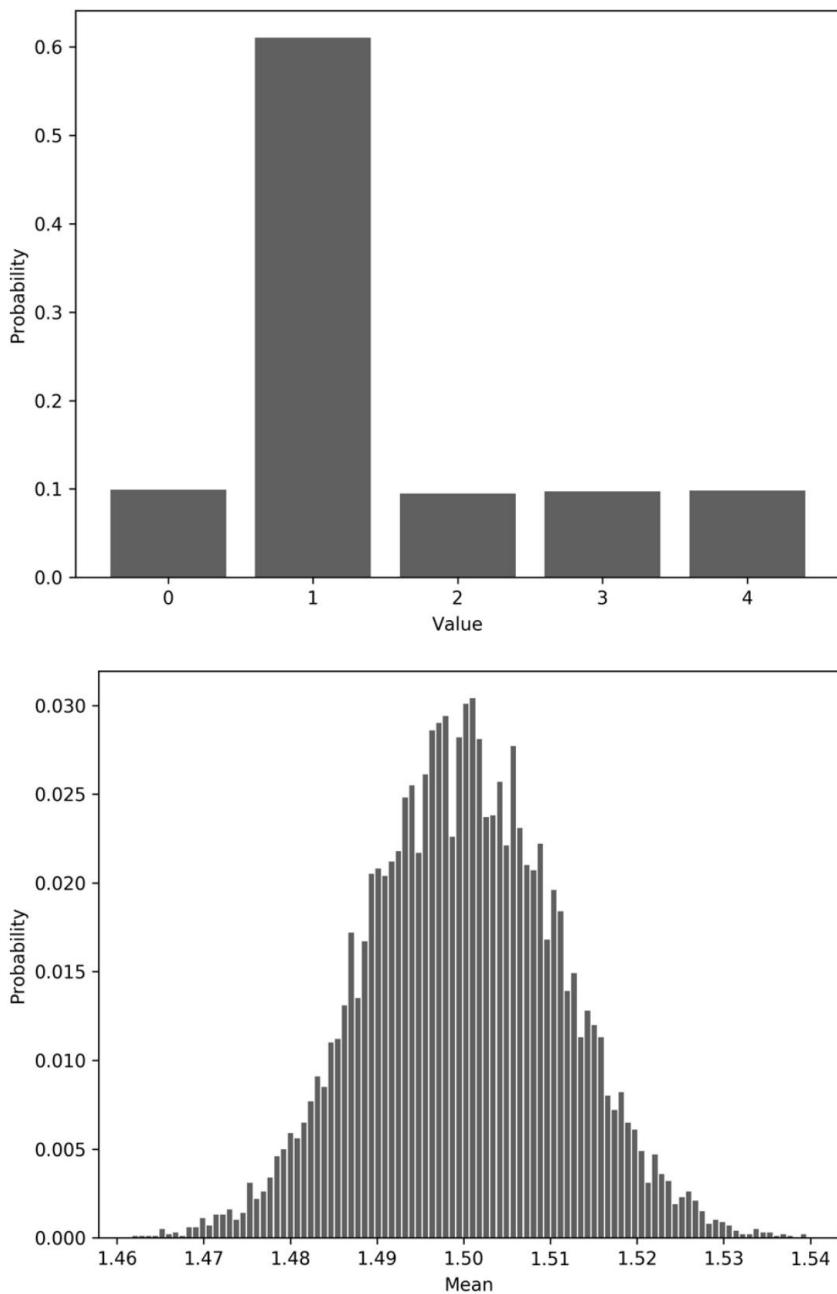


Figura 3-6: Una distribución discreta arbitraria (arriba) y la distribución de medias muestrales extraídas de ella (abajo)

### La ley de los grandes números

Un concepto relacionado con el teorema del límite central, y que a menudo se confunde con él, es la ley de los grandes números. La ley de los grandes números establece que a medida que aumenta el tamaño de una muestra de una distribución, la media de la muestra se acerca cada vez más a la media de la población. En este caso, estamos contemplando una sola muestra de la distribución y haciendo una declaración sobre qué tan cerca esperamos que esté su media de la verdadera media poblacional. Para el teorema del límite central, tenemos muchos conjuntos diferentes de muestras de la distribución y estamos haciendo una declaración sobre la distribución de las medias de esos conjuntos de muestras.

Podemos demostrar la ley de los números grandes simplemente seleccionando muestras cada vez más grandes de una distribución y rastreando la media en función del tamaño de la muestra (el número de muestras extraídas). En código, entonces,

---

```
m = []
para n en np.linspace(1,8,30):
    t = np.random.normal(1,1,size=int(10**n))
    m.append(t.mean())
```

---

donde estamos extrayendo tamaños de muestra cada vez más grandes de una distribución normal con una media de 1. El primer tamaño de muestra es 10 y el último es 100 millones. Si trazamos la media de las muestras en función del tamaño de la muestra, vemos la ley de los grandes números en acción.

La Figura 3-7 muestra las medias muestrales en función del número de muestras para la distribución normal con una media de 1 (línea discontinua). A medida que aumenta el número de muestras de la distribución, la media de las muestras se acerca a la media poblacional, lo que ilustra la ley de los grandes números.

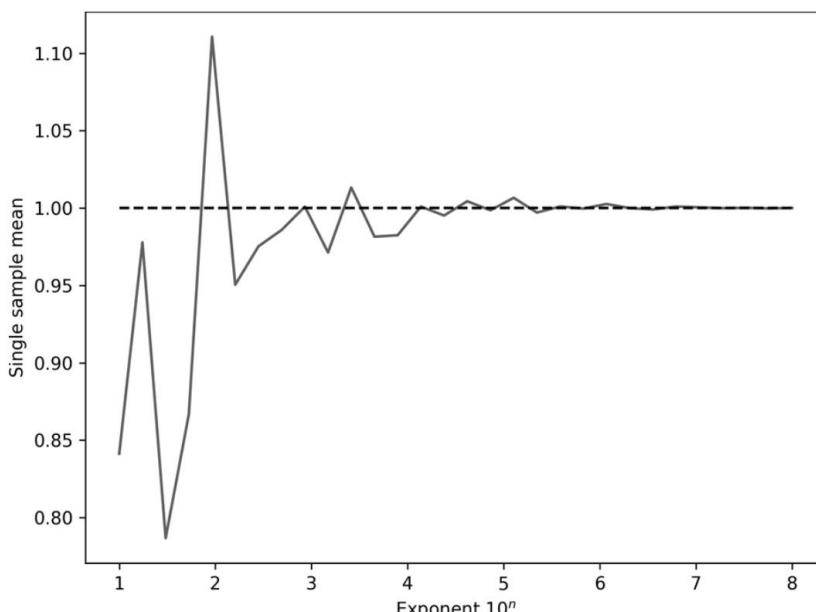


Figura 3-7: La ley de los grandes números en acción

Cambiemos de tema y pasemos al teorema de Bayes, el último tema de este capítulo.

## Teorema de Bayes

En el Capítulo 2, analizamos un ejemplo en el que determinamos si una mujer tenía cáncer. Allí, prometí que el teorema de Bayes nos diría cómo explicar adecuadamente la probabilidad de que una mujer de unos 40 años seleccionada al azar tuviera cáncer de mama. Cumplimos esa promesa en esta sección aprendiendo qué es el teorema de Bayes y cómo usarlo.

Usando la regla del producto, ecuación 2.8, conocemos las siguientes dos matemáticas: Las afirmaciones matemáticas son verdaderas:

$$P(B,A) = P(B|A)P(A)$$

$$P(A, B) = P(A|B)P(B)$$

Además, debido a que la probabilidad conjunta de A y B no depende de qué evento llamamos A y cuál llamamos B,

$$P(A,B) = P(B,A)$$

Por lo tanto,

$$P(B|A)P(A) = P(A|B)P(B)$$

Dividiendo por P(A), obtenemos

$$P(B|A) = \frac{P(A|B)P(B)}{\text{PENSILVANIA}} \quad (3.1)$$

Este es el teorema de Bayes, el corazón del enfoque bayesiano de la probabilidad y la forma adecuada de comparar dos probabilidades condicionales:  $P(B|A)$  y  $P(A|B)$ . A veces verás que la ecuación 3.1 se conoce como regla de Bayes. A menudo tampoco verás ningún apóstrofo después de "Bayes", que es un poco descuidado y agramatical, pero común.

La ecuación 3.1 ha quedado plasmada en luces de neón, tatuajes e incluso nombres de bebés: "Bayes". La ecuación lleva el nombre de Thomas Bayes (1701-1761), ministro y estadístico inglés, y se publicó después de su muerte. En palabras, la Ecuación 3.1 dice lo siguiente:

La probabilidad posterior,  $P(B|A)$ , es el producto de  $P(A|B)$ , la verosimilitud, y  $P(B)$ , la anterior, normalizada por  $P(A)$ , la probabilidad marginal o evidencia..

Ahora que sabemos qué es el teorema de Bayes, veámoslo en acción para poder entenderlo.

## Cáncer o no Redux

Una forma de pensar en los componentes del teorema de Bayes es en el contexto de las pruebas médicas. Al comienzo del capítulo 2, calculamos la probabilidad de que una mujer tuviera cáncer de mama con una mamografía positiva y descubrimos que era bastante diferente de lo que ingenuamente podríamos haber creído que era. Revisemos ahora ese problema usando el teorema de Bayes. Podría resultar útil volver a leer la primera sección del Capítulo 2 antes de continuar.

Queremos utilizar el teorema de Bayes para encontrar la probabilidad posterior, la probabilidad de cáncer de mama dada una mamografía positiva. Escribiremos esto como  $P(bc+|+) = P(+|bc+)$ , lo que significa cáncer de mama ( $bc+$ ) dada una mamografía positiva (+).

En el problema, se nos dice que la mamografía arroja un resultado positivo, dado que la paciente tiene cáncer de mama, el 90 por ciento de las veces. Escribimos esto como

$$P(+|bc+) = 0,9$$

Ésta es la probabilidad de una mamografía positiva en términos de la ecuación de Bayes,  $P(A|B) = P(+|bc+)$ .

A continuación, se nos dice que la probabilidad de que una mujer al azar tenga cáncer de mama es del 0,8 por ciento. Por lo tanto, sabemos

$$P(bc+) = 0,008$$

Ésta es la probabilidad previa,  $P(B)$ , en el teorema de Bayes.

Tenemos todos los componentes de la Ecuación 3.1 excepto uno:  $P(A)$ . ¿Qué es  $P(A)$  en este contexto? Es  $P(+)$ , la probabilidad marginal de una mamografía positiva independientemente de cualquier  $B$ , cualquier estado de cáncer de mama. También es la evidencia que tenemos, lo que sabemos: la mamografía fue positiva.

En el problema, se nos dice que hay un 7 por ciento de posibilidades de que una mujer sin cáncer de mama tenga una mamografía positiva. ¿Es esto  $P(+) = P(+|bc-)$ ? No, es  $P(+|bc-)$ , la probabilidad de una mamografía positiva si no hay cáncer de mama.

Ya me he referido dos veces a  $P(A)$  como probabilidad marginal. Sabemos qué hacer para obtener una probabilidad marginal o total: sumamos todas las demás partes de una probabilidad conjunta que no importan para lo que queremos saber. Aquí, tenemos que sumar todas las particiones del espacio muestral que no nos interesan para obtener la probabilidad marginal de una mamografía positiva. ¿Qué particiones son esas? Sólo hay dos: o una mujer tiene cáncer de mama o no. Por lo tanto, necesitamos encontrar

$$P(+) = P(+|bc+)P(bc+) + P(+|bc-)P(bc-)$$

Ya conocemos todas estas cantidades, excepto  $P(bc-)$ . Esta es la probabilidad previa de que una mujer seleccionada al azar no tenga cáncer de mama,  $P(bc-) = 1 - P(bc+) = 0,992$ .

A veces, verás la suma de otros términos en la probabilidad conjunta expresada en el denominador del teorema de Bayes. Incluso si no se mencionan explícitamente, están ahí, implícitos en lo que se necesita para encontrar  $P(A)$ .

Finalmente, tenemos todas las piezas y podemos calcular la probabilidad usando Teorema de Bayes:

$$\begin{aligned}
 P(\text{antes de Cristo} + |+) &= \frac{P(+|bc+)P(bc+)}{P(+|bc+)P(bc+) + P(+|bc-)P(bc-)} \\
 &= \frac{0,9(0,008)}{0,9(0,008) + 0,07(0,992)} \\
 &= 0,094 \approx 9 \text{ por ciento}
 \end{aligned}$$

Este es el resultado que encontramos anteriormente. Recordemos que un gran porcentaje de médicos en el estudio afirmaron que la probabilidad de cáncer a partir de una mamografía positiva,  $P(A|B)$ , era del 90 por ciento. Su error fue equiparar incorrectamente  $P(A|B)$  con  $P(B|A)$ . El teorema de Bayes relaciona correctamente los dos utilizando la probabilidad previa y marginal.

#### Actualización del

anterior No es necesario que nos detengamos con este único cálculo. Considere lo siguiente: ¿qué pasa si, después de que una mujer recibe la noticia de que su mamografía es positiva, decide hacerse una segunda mamografía en otro centro con diferentes radiólogos leyendo los resultados, y esa mamografía también resulta positiva? ¿Aún cree que su probabilidad de tener cáncer de mama es del 9 por ciento? Intuitivamente, podríamos pensar que ahora tiene más razones para creer que tiene cáncer. ¿Se puede cuantificar esta creencia? Puede, en la visión bayesiana, actualizar el anterior,  $P(bc+)$ , con el posterior calculado a partir de la primera prueba,  $P(bc+ |+)$ . Después de todo, ahora tiene una mayor probabilidad previa de cáncer dada la primera mamografía positiva.

Calculemos este nuevo posterior en función del resultado de la mamografía anterior:

$$\begin{aligned}
 P(\text{antes de Cristo} + |+) &= \frac{P(+|bc+)P(bc+)}{P(+|bc+)P(bc+) + P(+|bc-)P(bc-)} \\
 &= \frac{0,9(0,094)}{0,9(0,094) + 0,07(0,906)} \\
 &= 0,572 \approx 57 \text{ por ciento}
 \end{aligned}$$

Como el 57 por ciento es significativamente más alto que el 9 por ciento, nuestra mujer hipotética ahora tiene muchas más razones para creer que tiene cáncer de mama.

Observe lo que ha cambiado en este nuevo cálculo, además de un aumento dramático en la probabilidad posterior de cáncer de mama dado el resultado positivo de la segunda mamografía. Primero, la probabilidad previa de cáncer de mama disminuyó

de 0,008 → 0,094, el posterior calculado en base a la primera prueba. Segundo,  $P(bc^-)$  también cambió de 0,992 → 0,906. ¿Por qué? Porque el anterior cambió y  $P(bc^-) = 1 - P(bc^+)$ . La suma de  $P(bc^+)$  y  $P(bc^-)$  aún debe ser 1,0—O tiene cáncer de mama o no; ese es todo el espacio muestral.

En el ejemplo anterior, actualizamos el anterior en función del resultado de la prueba inicial y se nos proporcionó un anterior inicial en el primer ejemplo. Qué pasa el prior en general? En muchos casos, los bayesianos seleccionan lo prioritario, al menos inicialmente, basándose en una creencia real sobre el problema. A menudo, el prior es una distribución uniforme, conocida como prior desinformada porque no hay nada para guiar la selección de cualquier otra cosa. Para el ejemplo del cáncer de mama, el Prior es algo que se puede estimar a partir de un experimento que utiliza una selección aleatoria de mujeres de la población general.

Como se mencionó anteriormente, no se tome los números demasiado en serio; ellos son, por ejemplo, de uso exclusivo. Además, si bien una mujer ciertamente tiene la opción de obtener una segunda opinión, el estándar de oro para un diagnóstico de cáncer de mama es biopsia, el siguiente paso probable después de una mamografía inicial positiva. Finalmente, A lo largo de esta sección me he referido a las mujeres y el cáncer de mama. Los hombres también contraer cáncer de mama, aunque es poco común, con menos del 1 por ciento de los casos en hombres. Sin embargo, simplificó el debate al referirse únicamente a las mujeres. tomaré nota que los casos de cáncer de mama en hombres tienen más probabilidades de ser fatales, aunque aún no se conocen las razones.

### Teorema de Bayes en aprendizaje automático

El teorema de Bayes prevalece en todo el aprendizaje automático y el aprendizaje profundo. Un uso clásico del teorema de Bayes, que puede funcionar sorprendentemente bien, es usarlo como clasificador. Esto se conoce como clasificador Naive Bayes. Temprano Los filtros de spam de correo electrónico utilizaron este enfoque con bastante eficacia.

Supongamos que tenemos un conjunto de datos que consta de etiquetas de clase,  $y$ , y vectores de características,  $x$ . El objetivo de un clasificador Naive Bayes es decirnos, para cada clase, el probabilidad de que un vector de características dado pertenezca a esa clase. Con esas probabilidades, podemos asignar una etiqueta de clase seleccionando la probabilidad más grande. Eso es decir, queremos encontrar  $P(y|x)$  para cada etiqueta de clase,  $y$ . Esta es una probabilidad condicional, por lo que podemos usar el teorema de Bayes con ella:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \quad (3.2)$$

La ecuación anterior dice que la probabilidad de que el vector de características  $x$  represente una instancia de la etiqueta de clase  $y$  es la probabilidad de que la etiqueta de clase  $y$  genere un vector de características  $x$  multiplicada por la probabilidad previa de que ocurra la etiqueta de clase  $y$ , dividida por la probabilidad marginal del vector de características sobre todas las clases etiquetas. Recuerde la suma implícita al calcular  $P(x)$ .

¿En qué nos resulta útil esto? Como tenemos un conjunto de datos, podemos estimar  $P(y)$  usándolo, asumiendo que la distribución de clases del conjunto de datos es una representación justa de lo que encontraríamos al usar el modelo. Y, como tenemos etiquetas, Puede dividir el conjunto de datos en colecciones más pequeñas, por clase. Esto podría ayudar

Hagamos algo útil para obtener las probabilidades por clase,  $P(x|y)$ . Ignoraremos completamente el  $P(x)$  marginal . Veamos por qué, en este caso, somos libres de hacerlo.

La ecuación 3.2 es para una etiqueta de clase particular, digamos  $y = 1$ . Tendremos otras versiones para todas las etiquetas de clase en el conjunto de datos. Dijimos que nuestro clasificador consiste en calcular las probabilidades posteriores para cada etiqueta de clase y seleccionar la más grande como etiqueta asignada a un vector de características desconocido. El denominador de la Ecuación 3.2 es un factor de escala, que hace que el resultado sea una probabilidad real. Sin embargo, para nuestro caso de uso, solo nos preocupamos por el orden relativo de  $P(y|x)$  entre las diferentes etiquetas de clase. Dado que  $P(x)$  es el mismo para todo  $y$ , es un factor común que cambiará el número asociado con  $P(y|x)$  pero no el orden de las diferentes etiquetas de clase. Por lo tanto, podemos ignorarlo y concentrarnos en encontrar los productos de las probabilidades y los antecedentes.

Aunque el  $P(y|x)$  más grande calculado de esta manera ya no es una probabilidad adecuada, sigue siendo la etiqueta de clase correcta para asignar.

Dado que podemos ignorar  $P(x)$  y que los valores de  $P(y)$  se estiman fácilmente a partir del conjunto de datos, nos queda calcular  $P(x|y)$ , la probabilidad de que dada la etiqueta de clase sea  $y$ , tendríamos un vector de características  $x$ . ¿Qué podemos hacer en este caso?

Primero, podemos pensar en qué es  $P(x|y)$ . Es una probabilidad condicional para los vectores de características dado que todos los vectores de características son representantes de la clase  $y$ . Por el momento, ignoremos la parte  $y$ , ya que sabemos que todos los vectores de características provienen de la clase  $y$ .

Esto deja solo  $P(x)$  porque fijamos  $y$ . Un vector de características es una colección de características individuales,  $x = (x_0, x_1, x_2, \dots, x_{n-1})$  para  $n$  características en el vector. Por lo tanto,  $P(x)$  es en realidad una probabilidad conjunta, la probabilidad de que todas las características individuales tengan sus valores específicos al mismo tiempo. Entonces, podemos escribir

$$P(x) = P(x_0, x_1, x_2, \dots, x_{n-1})$$

¿Cómo ayuda esto? Si hacemos una suposición más sobre nuestros datos, Veremos que podemos descomponer esta probabilidad conjunta de una manera conveniente. Supongamos que todas las características de nuestro vector de características son independientes. Recuerde que independiente significa que el valor de  $x_1$  , por ejemplo, no se ve afectado de ninguna manera por el valor de ninguna otra característica del vector. Por lo general, esto no es del todo cierto, y para cosas como los píxeles en las imágenes definitivamente no es cierto, pero asumiremos que es cierto de todos modos. Somos ingenuos al creer que es verdad, de ahí el Naive en Naive Bayes.

Si las características son independientes, entonces la probabilidad de que una característica tome cualquier valor particular es independiente de todas las demás. En ese caso, la regla del producto nos dice que podemos dividir la probabilidad conjunta así:

$$P(x) = P(x_0)P(x_1)P(x_2)\dots P(x_{n-1})$$

Esto ayuda enormemente. Disponemos de un conjunto de datos, etiquetados por clase, que nos permite estimar la probabilidad de cualquier característica para cualquier clase específica contando con qué frecuencia ocurre cada valor de característica para cada clase.

Juntémoslo todo para obtener un conjunto de datos hipotético de tres clases (0, 1 y 2) y cuatro características. Primero utilizamos el conjunto de datos, dividido por etiqueta de clase, para estimar la probabilidad de cada valor de característica. Esto nos proporciona el conjunto de  $P(x_0)$ ,  $P(x_1)$ , etc., para cada característica de cada etiqueta de clase. Conjunto

con la probabilidad previa de la etiqueta de clase, estimada a partir del conjunto de datos como el número de cada clase dividido por el número total de muestras en el conjunto de datos, calculamos un nuevo vector de características desconocido,  $x$ ,

$$P(0|x) = P(x|0)P(0)$$

$$= P(x_0)P(x_1)P(x_2)P(x_3)P(0)$$

Aquí, las probabilidades de característica  $P(x_0)$  son específicas de la clase 0 únicamente, y  $P(0)$  es la probabilidad previa de la clase 0 en el conjunto de datos.  $P(0|x)$  es la probabilidad posterior no normalizada de que el vector de características desconocido  $x$  pertenezca a la clase 0. Decimos no normalizado porque estamos ignorando el denominador del teorema de Bayes, sabiendo que incluirlo no cambiaría el orden de las probabilidades posteriores, solo sus valores.

Podemos repetir el cálculo anterior para obtener  $P(1|x)$  y  $P(2|x)$ , haciendo asegúrese de utilizar las probabilidades por característica calculadas para esas clases (las  $P(x_0)$ ). Finalmente, le damos a  $x$  la etiqueta de clase para el mayor de los tres posteriores calculados.

La descripción anterior supone que los valores de las características son discretos. Por lo general, no lo son, pero existen soluciones. Una es agrupar los valores de las características para hacerlos discretos. Por ejemplo, si la característica está por encima de [0, 3], cree una nueva característica que sea 0, 1 o 2 y asigne la característica continua a uno de esos contenedores truncando cualquier parte fraccionaria.

Otra solución es hacer una suposición más sobre la distribución de la que provienen los valores de las características y usar esa distribución para calcular los  $P(x_0)$  por clase. Las características a menudo se basan en mediciones del mundo real, y muchas cosas en el mundo real siguen una distribución normal.

Por lo tanto, normalmente asumiríamos que las características individuales, aunque continuas, están distribuidas normalmente y podemos encontrar estimaciones de la media ( $\mu$ ) y la desviación estándar ( $\sigma$ ) del conjunto de datos, por característica y etiqueta de clase.

El teorema de Bayes es útil para calcular probabilidades. Es útil en machine learning también. La batalla entre bayesianos y frecuentistas parece estar menguando, aunque persisten diferencias filosóficas. En la práctica, la mayoría de los investigadores están aprendiendo que ambos enfoques son valiosos y, en ocasiones, se deben utilizar herramientas de ambos campos. Continuaremos esta tendencia en el próximo capítulo, donde examinaremos las estadísticas desde un punto de vista frecuentista. Defendemos esta decisión señalando que la gran mayoría de los resultados científicos publicados en el último siglo utilizaron las estadísticas de esta manera, lo que incluye a la comunidad de aprendizaje profundo, al menos cuando presenta los resultados de los experimentos.

## Resumen

Este capítulo nos enseñó sobre las distribuciones de probabilidad, qué son y cómo extraer muestras de ellos, tanto discretas como continuas. Encontraremos diferentes distribuciones durante nuestra exploración del aprendizaje profundo. Nosotros También descubrió el teorema de Bayes y vio cómo nos permite relacionar adecuadamente probabilidades condicionales. Vimos cómo el teorema de Bayes nos permite evaluar la verdadera probabilidad de cáncer dada una prueba médica imperfecta: una situación común. También aprendimos a utilizar el teorema de Bayes, junto con algunos de los reglas básicas de probabilidad que aprendimos en el Capítulo 2, para construir un sistema simple pero a menudo Clasificador sorprendentemente eficaz.

Pasemos ahora al mundo de la estadística.



# 4

## ESTADÍSTICAS



Los conjuntos de datos incorrectos conducen a malos modelos. Nos gustaría comprender nuestros datos antes de construir un modelo y luego usar esa comprensión para crear un conjunto de datos útil, uno que conduzca a modelos que hagan lo que esperamos que hagan. Conocer las estadísticas básicas nos permitirá hacer precisamente eso.

Una estadística es cualquier número que se calcula a partir de una muestra y se utiliza para caracterizarla de alguna manera. En el aprendizaje profundo, cuando hablamos de muestras, normalmente nos referimos a conjuntos de datos. Quizás la estadística más básica sea la media aritmética, comúnmente conocida como promedio. La media de un conjunto de datos es un resumen de un solo número del conjunto de datos.

Veremos muchas estadísticas diferentes en este capítulo. Comenzaremos aprendiendo sobre los tipos de datos y caracterizando un conjunto de datos con estadísticas resumidas. A continuación, aprenderemos sobre cuantiles y cómo trazar datos para comprender lo que contienen. Después viene una discusión sobre los valores atípicos y los datos faltantes. Los conjuntos de datos rara vez son perfectos, por lo que necesitamos tener alguna forma de detectar datos incorrectos y lidiar con los datos faltantes. Continuaremos nuestra discusión sobre conjuntos de datos imperfectos con una discusión sobre la correlación entre variables. Luego cerraremos el capítulo analizando las pruebas de hipótesis, donde intentaremos responder preguntas como "¿Qué probabilidad hay de que el mismo proceso principal genere dos conjuntos de datos?" Las pruebas de hipótesis se utilizan ampliamente en la ciencia, incluido el aprendizaje profundo.

## Tipos de datos

Los cuatro tipos de datos son nominales, ordinales, de intervalo y de razón. Veamos cada uno por separado.

### Datos nominales

Los datos nominales, a veces llamados datos categóricos, son datos que no tienen ningún orden entre los diferentes valores. Un ejemplo de esto es el color de ojos; No existe ninguna relación entre el marrón, el azul y el verde.

### Datos ordinales

Para los datos ordinales, los datos tienen una clasificación u orden, aunque las diferencias no son significativas en un sentido matemático. Por ejemplo, si un cuestionario le pide que seleccione entre "totalmente en desacuerdo", "en desacuerdo", "neutral", "de acuerdo" y "totalmente de acuerdo", está bastante claro que hay un orden. Aún así, también está claro que "de acuerdo" no es más que "totalmente en desacuerdo". Todo lo que podemos decir es que "totalmente en desacuerdo" está a la izquierda de "de acuerdo" (y "neutral" y "en desacuerdo").

Otro ejemplo de datos ordinales es el nivel de educación. Si una persona tiene educación de cuarto grado y otra tiene educación de octavo grado, podemos decir que esta última tiene más educación que la primera, pero no podemos decir que esta última tiene el doble de educación, porque "el doble de como educado" no tiene un significado fijo.

### Datos de intervalo

Los datos de intervalo tienen diferencias significativas. Por ejemplo, si un vaso de agua está a 40 grados Fahrenheit y otro a 80 grados Fahrenheit, podemos decir que hay una diferencia de 40 grados entre los dos vasos de agua. Sin embargo, no podemos decir que hay el doble de calor en la segunda taza, porque el cero de la escala Fahrenheit es arbitrario. Coloquialmente decimos que hace el doble de calor, pero en realidad no lo es. Para ver esto, pensemos en lo que sucede si cambiamos la escala de temperatura a otra escala con un cero arbitrario, aunque más sensato: la escala Celsius. Vemos que la primera taza está a unos 4,4 grados Celsius y la segunda a 26,7 grados Celsius. Está claro que la segunda taza no tiene ahora de repente seis veces más picante que la primera.

### Datos de relación

Finalmente, los datos de ratios son datos en los que las diferencias son significativas y hay un verdadero punto cero. La altura es un valor de razón porque una altura de cero es simplemente eso: ninguna altura en absoluto. De manera similar, la edad también es un valor de proporción porque una edad de cero significa que no hay edad alguna. Si adoptáramos una nueva escala de edad y llamáramos cero a una persona cuando alcance, digamos, la edad para votar, entonces tendríamos una escala de intervalo, no una escala de razón.

Miremos la temperatura nuevamente. Dijimos anteriormente que la temperatura es una cantidad de intervalo. Este no es siempre el caso. Si medimos la temperatura en

Fahrenheit o Celsius, entonces sí, es una cantidad de intervalo. Sin embargo, si medimos la temperatura en Kelvin, la escala de temperatura absoluta, entonces se convierte en un valor de relación. ¿Por qué? Porque una temperatura de 0 Kelvin (o K) es simplemente eso, ninguna temperatura. Si nuestra primera taza está a 40°F, 277.59 K, y la segunda está a 80°F, 299.82 K, entonces podemos decir con certeza que la segunda taza está 1.08 veces más caliente que la primera, ya que  $(277.59)(1.08) \approx 299.8$ .

La Figura 4-1 nombra las escalas y muestra sus relaciones entre sí.

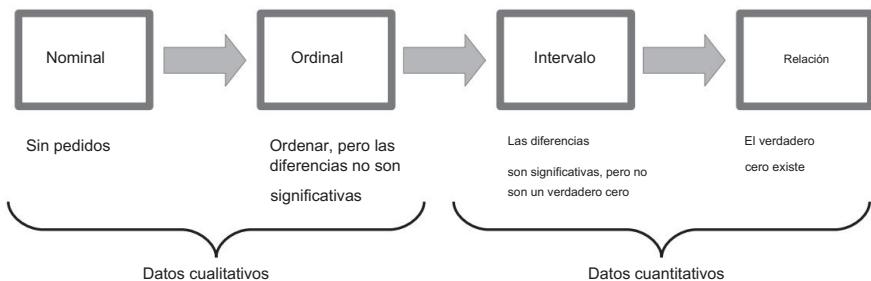


Figura 4-1: Los cuatro tipos de datos

Cada paso en la Figura 4-1 de izquierda a derecha agrega algo a los datos que falta el tipo de datos de la izquierda. De nominal a ordinal, agregamos orden. Para ordinal a intervalo, agregamos diferencias significativas. Por último, pasar del intervalo al ratio añade un verdadero punto cero.

En el uso práctico, en lo que a estadísticas se refiere, debemos ser conscientes de los tipos de datos para no hacer algo sin sentido. Si tenemos un cuestionario y el valor medio de la pregunta A en una escala de calificación de 1 a 5 es 2, mientras que para la pregunta B es 4, no podemos decir que B tiene una calificación dos veces mayor que A, solo que B obtuvo una calificación superior a A. Lo que significa "dos veces" en este contexto no está claro y probablemente no tenga sentido.

Los datos de intervalos y razones pueden ser continuos (puntos flotantes) o discretos (enteros). Desde una perspectiva de aprendizaje profundo, los modelos suelen tratar los datos continuos y discretos de la misma manera, y no necesitamos hacer nada especial con los datos discretos.

#### Uso de datos nominales en aprendizaje

profundo Si tenemos un valor nominal, digamos un conjunto de colores, como rojo, verde y azul, y queremos pasar ese valor a una red profunda, necesitamos cambiar los datos antes de poder usarlos. él. Como acabamos de ver, los datos nominales no tienen orden, por lo que si bien es tentador asignar un valor de 1 al rojo, 2 al verde y 3 al azul, sería un error hacerlo, ya que la red interpretará esos números como intervalo. datos. En ese caso, para la red, azul = 3(rojo), lo cual, por supuesto, no tiene sentido. Si queremos utilizar datos nominales con una red profunda, debemos modificarlos para que el intervalo sea significativo. Hacemos esto con codificación one-hot.

En la codificación one-hot, convertimos la variable nominal única en un vector, donde cada elemento del vector corresponde a uno de los valores nominales.

Para el ejemplo del color, la variable nominal se convierte en un vector de tres elementos: un elemento representa el rojo, otro el verde y el último el azul.

Luego, ponemos el valor correspondiente al color a uno y todos los demás a cero, así:

Valor	Vector
rojo → 1 0 0	verde
→ 0 1 0	azul → 0 0 1

Ahora los valores del vector son significativos porque es rojo (1) o no es (0), verde (1) o no es (0), o azul (1) o no es (0). El intervalo entre cero y uno tiene significado matemático porque la presencia del valor, digamos rojo, es realmente mayor que su ausencia, y eso funciona de la misma manera para cada color. Los valores ahora son de intervalo, por lo que la red puede usarlos.

En algunos kits de herramientas, como Keras, las etiquetas de clase se codifican en caliente antes de pasárselas al modelo. Esto se hace para que la salida vectorial funcione bien con la etiqueta de clase codificada en caliente al calcular la función de pérdida.

## Resumen estadístico

Nos dan un conjunto de datos. ¿Cómo le damos sentido? ¿Cómo deberíamos caracterizarlo para comprenderlo mejor antes de usarlo para construir un modelo?

Para responder a estas preguntas, necesitamos aprender sobre estadísticas resumidas. Calcular estadísticas resumidas debería ser lo primero que haga cuando entregue un nuevo conjunto de datos. No mirar su conjunto de datos antes de construir un modelo es como comprar un automóvil usado sin revisar los neumáticos, probarlo y mirar debajo del capó.

Las personas tienen diferentes nociones sobre lo que constituye un buen conjunto de estadísticas resumidas. Nos centraremos en lo siguiente: medios; la mediana; y medidas de variación, incluida la varianza, la desviación estándar y el error estándar.

El alcance y el modo también se mencionan a menudo. El rango es la diferencia entre el máximo y el mínimo del conjunto de datos. La moda es el valor más frecuente en el conjunto de datos.

Generalmente tenemos una idea visual de la moda a partir del histograma, ya que el histograma nos muestra la forma de la distribución de los datos.

## Medias y mediana

La mayoría de nosotros aprendimos a calcular el promedio de un conjunto de números en la escuela primaria: sumar los números y dividirlos por cuántos hay. Ésta es la media aritmética o, más específicamente, la media aritmética no ponderada.

Si el conjunto de datos consta de un conjunto de valores,  $\{x_0, x_1, x_2, \dots, x_{n-1}\}$ , entonces la media aritmética es la suma de los datos dividida por el número de elementos en el conjunto de datos ( $n$ ). Notacionalmente, escribimos esto de la siguiente manera.

$$x = \frac{1}{n} \sum_{i=1}^{n-1} x_i \quad (4.1)$$

La  $x$  es la forma típica de indicar la media de una muestra.

La ecuación 4.1 calcula la media no ponderada. A cada valor se le asigna un peso de  $1/n$ , donde la suma de todos los pesos es 1,0. A veces, es posible que queramos ponderar los elementos del conjunto de datos de manera diferente; en otras palabras, no todos deberían contar por igual. En ese caso, calculamos una media ponderada,

$$x = \frac{\sum_{i=1}^{n-1} w_i x_i}{\sum_{i=1}^{n-1} w_i}$$

donde  $w_i$  es el peso dado a  $x_i$  y  $\sum_i w_i = 1$ . Los pesos no forman parte del conjunto de datos; necesitan venir de algún otro lugar. El promedio de calificaciones (GPA) utilizado por muchas universidades es un ejemplo de media ponderada.

La calificación de cada curso se multiplica por el número de créditos del curso y la suma se divide por el número total de créditos. Algebraicamente, esto equivale a multiplicar cada calificación por un peso,  $w_i = c_i / \sum_i c_i$ , siendo  $c_i$  el número de créditos para el curso  $i$  y  $\sum_i c_i$  el número total de créditos para el curso semestre.

#### Significado geométrico

La media aritmética es, con diferencia, la media más utilizada. Sin embargo, hay otros. La media geométrica de dos números positivos,  $a$  y  $b$ , es la raíz cuadrada de su producto:

$$x_g = \sqrt{ab}$$

En general, la media geométrica de  $n$  números positivos es la raíz enésima de su producto:

$$x_g = \sqrt[n]{x_0 x_1 x_2 \dots x_{n-1}}$$

La media geométrica se utiliza en finanzas para calcular las tasas de crecimiento promedio. En el procesamiento de imágenes, la media geométrica se puede utilizar como filtro para ayudar a reducir el ruido de la imagen. En el aprendizaje profundo, la media geométrica aparece en el coeficiente de correlación de Matthews (MCC), una de las métricas que utilizamos para evaluar los modelos de aprendizaje profundo. El MCC es la media geométrica de otras dos métricas, la información y el marcado.

#### Significado armónico

La media armónica de dos números,  $a$  y  $b$ , es el recíproco de la media aritmética de sus recíprocos:

$$x_h = \frac{1}{\frac{1}{a} + \frac{1}{b}} - 1$$

En general,

$$x_h = \left( 1 - \sum_{i=1}^{n-1} \frac{1}{x_i} \right)^{-1} = \frac{1}{x_0} + \frac{1}{x_1} + \dots + \frac{1}{x_{n-1}}$$

La media armónica aparece en el aprendizaje profundo como la puntuación F1. Esta es una métrica de uso frecuente para evaluar clasificadores. La puntuación F1 es la media armónica de la recuperación (sensibilidad) y la precisión:

$$F1 = \left( 1 - \frac{1}{\frac{1}{\text{recordar}} + \frac{1}{\text{precisión}}} \right)^{-1}$$

$$= 2 \frac{\text{recordar} \cdot \text{precisión}}{\text{recordar} + \text{precisión}}$$

A pesar de su uso frecuente, no es una buena idea utilizar la puntuación F1 para evaluar un modelo de aprendizaje profundo. Para ver esto, considere las definiciones de recuperación y precisión:

$$\text{recordar} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Aquí, TP es el número de verdaderos positivos, FN es el número de falsos negativos y FP es el número de falsos positivos. Estos valores provienen del conjunto de pruebas utilizado para evaluar el modelo. Un cuarto número que es importante para los clasificadores, TN, es el número de verdaderos negativos clasificados correctamente (suponiendo un clasificador binario). La puntuación F1 ignora TN, pero para comprender qué tan bien se desempeña el modelo, debemos considerar clasificaciones tanto positivas como negativas. Por tanto, la puntuación de la F1 es engañosa y, a menudo, demasiado optimista. Mejores métricas son el MCC mencionado anteriormente o el de Cohen (kappa), que es similar al MCC y generalmente lo sigue de cerca.

#### Mediana

Antes de pasar a las medidas de variación, hay una estadística resumida más utilizada que mencionaremos aquí. También aparecerá de nuevo un poco más adelante en el capítulo. La mediana de un conjunto de datos es el valor medio. Es el valor donde, cuando el conjunto de datos se ordena numéricamente, la mitad de los valores están debajo y la otra mitad encima. Usemos este conjunto de datos:

$$X = \{55, 63, 65, 37, 74, 71, 73, 87, 69, 44\}$$

Si ordenamos X, obtenemos

$$\{37, 44, 55, 63, 65, 69, 71, 73, 74, 87\}$$

Inmediatamente vemos un problema potencial. Dije que necesitamos el valor medio cuando se ordenan los datos. Con 10 cosas en X, no hay un valor medio. El medio se encuentra entre 65 y 69. Cuando el número de elementos del conjunto de datos es par, la mediana es la media aritmética de los dos números del medio.

Por lo tanto, la mediana en este caso es

$$x_{\text{mediana}} = \frac{65 + 69}{2} = 67$$

La media aritmética de los datos es 63,8. Cuál es la diferencia entre la media y la mediana?

Por diseño, la mediana nos indica el valor que divide el conjunto de datos, por lo que el número de muestras de arriba es igual al número de abajo. Lo que importa es el número de muestras. Para la media, es una suma de los valores de datos reales. Por lo tanto, la media es sensible a los valores mismos, mientras que la mediana es sensible al orden de los valores.

Si miramos X, vemos que la mayoría de los valores están entre 60 y 70, con un valor bajo de 37. Es el valor bajo de 37 el que arrastra la media hacia abajo en relación con la mediana. Un excelente ejemplo de este efecto son los ingresos. El ingreso familiar anual medio actual en Estados Unidos es de unos 62.000 dólares. Una medida reciente del ingreso familiar medio en los Estados Unidos se acerca más a los 72.000 dólares.

La diferencia se debe a que una pequeña porción de la población gana significativamente más dinero que los demás. Aumentan la media general.

Entonces, para el ingreso, la estadística más significativa es la mediana.

Considere la Figura 4-2.

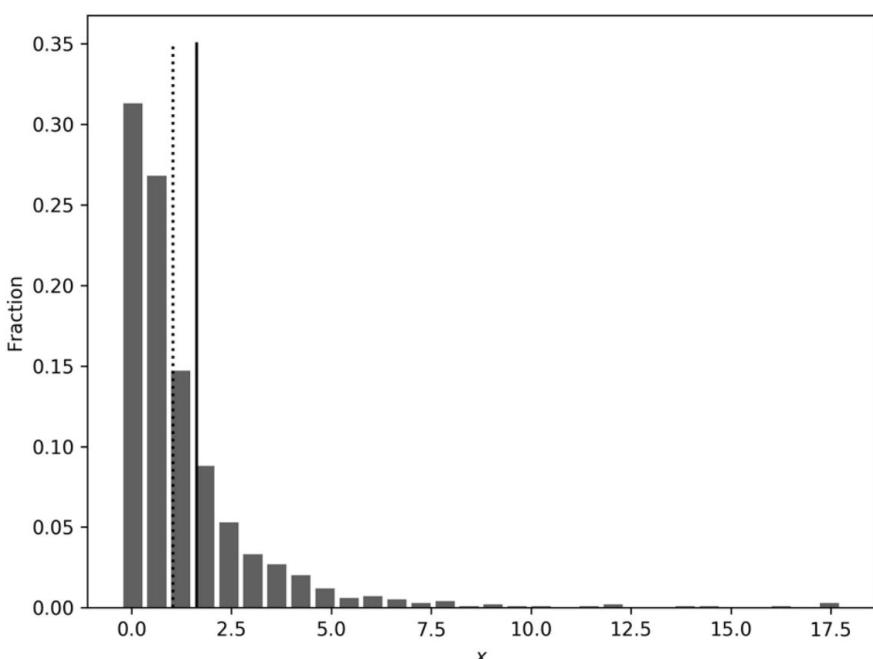


Figura 4-2: La media (sólida) y la mediana (discontinua) trazadas sobre el histograma de un conjunto de datos de muestra

La Figura 4-2 muestra el histograma generado a partir de 1000 muestras de un conjunto de datos simulado. También se representan la media (línea continua) y la mediana (línea discontinua). Los dos no coinciden; la cola larga en el histograma arrastra la media hacia arriba. Si tuviéramos que contar, 500 muestras caerían en los contenedores debajo de la línea discontinua y 500 en los contenedores superiores.

¿Hay momentos en que la media y la mediana son iguales? Sí. Si la distribución de los datos es completamente simétrica, entonces la media y la mediana serán iguales. El ejemplo clásico de esta situación es la distribución normal.

La Figura 3-4 mostró una distribución normal donde la simetría izquierda-derecha era clara. La distribución normal es especial. Lo veremos nuevamente a lo largo del capítulo. Por ahora, recuerde que cuanto más se acerque la distribución del conjunto de datos a una distribución normal, más cercanas estarán la media y la mediana.

También vale la pena recordar lo contrario: si la distribución del conjunto de datos está lejos de ser normal, como en la Figura 4-2, entonces la mediana probablemente sea la mejor estadística a considerar al resumir los datos.

## Medidas de variación

Un arquero principiante dispara 10 flechas a un objetivo. Ocho de las flechas del principiante dan en el blanco, dos fallan por completo y las ocho que dan en el blanco se distribuyen uniformemente a lo largo de él. Un arquero experto dispara 10 flechas a un blanco.

Todas las flechas del experto dieron en el blanco a unos pocos centímetros del centro. Piensa en la posición media de las flechas. Para el experto, todas las flechas están cerca del centro del objetivo, por lo que podemos ver que la posición media de las flechas estará cerca del centro. Para el principiante, ninguna de las flechas está cerca del centro del objetivo, pero están dispersas más o menos igualmente a la izquierda y a la derecha o encima y debajo del centro. Debido a esto, la posición promedio se equilibrará y también estará cerca del centro del objetivo.

Sin embargo, las flechas del primer arquero están dispersas; su ubicación varía mucho. Las flechas del segundo arquero, por otro lado, están muy agrupadas y hay poca variación en su posición. Una forma significativa de resumir y comprender un conjunto de datos es cuantificar su variación. Veamos cómo podemos hacer esto.

### Desviación versus varianza

Una forma de medir la variación de un conjunto de datos es encontrar el rango, la diferencia entre los valores más grandes y más pequeños. Sin embargo, el rango es una medida burda, ya que no presta atención a la mayoría de los valores del conjunto de datos, solo a los extremos. Podemos hacerlo mejor calculando la media de la diferencia entre los valores de los datos y la media de los datos. La fórmula es

$$MD = \frac{1}{n} \sum_{i=1}^{n-1} |x_i - \bar{x}| \quad (4.2)$$

La ecuación 4.2 es la desviación media. Es una medida natural y da justo lo que queremos: una idea de qué tan lejos, en promedio, está cada muestra de la media.

Si bien no hay nada malo en calcular la desviación media, encontrarás

que rara vez se utiliza en la práctica. Una razón tiene que ver con el álgebra y el cálculo. Es molesto tratar matemáticamente el valor absoluto.

En lugar de la medida natural de variación, calculemos ésta usando diferencias al cuadrado:

$$\text{varianza} = \frac{1}{n-1} \sum_{i=1}^{n-1} (x_i - \bar{x})^2 \quad (4.3)$$

La ecuación 4.3 se conoce como varianza muestral sesgada. Es la media de la diferencia al cuadrado entre cada valor del conjunto de datos y la media. Es una forma alternativa de caracterizar la dispersión en el conjunto de datos. Por qué es parcial, lo discutiremos en un segundo. Poco después veremos por qué es  $s^2$  y no  $s$ .

Antes de hacerlo, vale la pena señalar que a menudo verás una ecuación ligeramente diferente:

$$\text{varianza} = \frac{1}{n-1} \sum_{i=1}^{n-1} (x_i - \bar{x})^2 \quad (4.4)$$

Esta ecuación es la varianza muestral insesgada. Usar  $n-1$  en lugar de  $n$  se conoce como corrección de Bessel. Está relacionado con el número de grados de libertad en los residuos, donde los residuos son lo que queda cuando se resta la media de cada uno de los valores del conjunto de datos. La suma de los residuos es cero, por lo que si hay  $n$  valores en el conjunto de datos, conocer  $n-1$  de los residuos permite calcular el último residuo. Esto nos da los grados de libertad de los residuos. Somos "libres" de calcular  $n-1$  de ellos, sabiendo que obtendremos el último por el hecho de que la suma de los residuos es cero. Dividir por  $n-1$  da una estimación menos sesgada de la varianza, suponiendo que, para empezar,  $s^2$  está sesgado de alguna manera.

¿Por qué hablamos de varianza sesgada y varianza insesgada?

¿Sesgado cómo? Siempre debemos recordar que un conjunto de datos es una muestra de algún proceso generador de datos principal: la población. La varianza poblacional verdadera ( $\sigma^2$ ) es la dispersión de la población alrededor de la media poblacional verdadera ( $\mu$ ). Sin embargo, no lo sabemos, sino que los estimamos a partir del conjunto de datos que tenemos. La media de la muestra es  $\bar{x}$ . Ésa es nuestra estimación para  $\mu$ . Entonces es natural calcular la media de las desviaciones al cuadrado alrededor de  $\bar{x}$  y llamar a eso nuestra estimación para  $\sigma^2$ . Eso es  $s^2$  (Ecuación 4.3).

La afirmación, que es cierta pero está más allá de nuestro alcance para demostrarla, es  $s^2$  sesgada y no es la mejor estimación de  $\sigma^2$ , pero si se aplica la corrección de Bessel,  $s^2$  tendremos una mejor estimación de la varianza poblacional. Entonces deberíamos usar  $s^2$  (Ecuación 4.4) para caracterizar la varianza del conjunto de datos alrededor de la media.

En resumen, deberíamos usar el conjunto  $\bar{x}$  para cuantificar la varianza de los datos  $x$  ands. Ahora bien,  $s^2$ ? La raíz cuadrada de la varianza es la desviación estándar.

¿Por qué se denota  $s$  para la población y  $s$  para la estimación calculada a partir del conjunto de datos? La mayoría de las veces queremos trabajar con la desviación estándar.

Escribir raíces cuadradas se vuelve tedioso, por lo que la convención ha adoptado la notación  $s$  para la desviación estándar y usa la forma cuadrada cuando se analiza la varianza.

Y, debido a que la vida aún no es lo suficientemente ambigua, a menudo verás que se usa para  $s$  y que se usa la Ecuación 4.3 cuando en realidad debería ser la Ecuación 4.4. Algunos kits de herramientas, incluido nuestro querido NumPy, facilitan el uso de la fórmula incorrecta.

Sin embargo, a medida que aumenta el número de muestras en nuestro conjunto de datos, la diferencia entre la varianza sesgada e insesgada disminuye porque dividir entre  $n$  o  $n - 1$  importa cada vez menos. Unas pocas líneas de código ilustran esto:

---

```
>>> importar números como np
>>> norte = 10
>>> a = np.random.random(n) >>> (1/
n)*(aa.media()**2).sum() 0.08081748204006689

>>> (1/(n-1))*(aa.media()**2).sum()
0.08979720226674098
```

---

Aquí, una muestra con solo 10 valores ( $a$ ) muestra una diferencia en la varianza sesgada e insesgada en el tercer decimal. Si aumentamos el tamaño de nuestro conjunto de datos de 10 a 10.000, obtenemos

---

```
>>> norte = 10000
>>> a = np.random.random(n) >>> (1/
n)*(aa.media()**2).sum() 0.08304350577482553

>>> (1/(n-1))*(aa.media()**2).sum()
0.08305181095592111
```

---

La diferencia entre la estimación sesgada e insesgada de la varianza está ahora en el quinto decimal. Por lo tanto, para los grandes conjuntos de datos con los que normalmente trabajamos en el aprendizaje profundo, en la práctica importa poco si usamos  $s_n$  o  $s$  para la desviación estándar.

#### DESVIACIÓN ABSOLUTA MEDIA

La desviación estándar se basa en la media. La media, como vimos anteriormente, es sensible a los valores extremos, y la desviación estándar lo es doblemente porque elevamos al cuadrado la desviación de la media para cada muestra. Una medida de variabilidad que es insensible a los valores extremos en el conjunto de datos es la desviación absoluta mediana (MAD). La DMA se define como la mediana de los valores absolutos de la diferencia entre los datos y la mediana:

$$\text{MAD} = \text{mediana}(|x - \text{mediana}(x)|)$$

De manera procesal, primero calcule la mediana de los datos, luego resta la media de cada valor de datos, haciendo que el resultado sea positivo, y reporte la mediana de ese conjunto. La implementación es sencilla:

---

```
def MAD(x):
    devuelve np.median(np.abs(x-np.median(x)))
```

---

El MAD no se utiliza con frecuencia, pero su insensibilidad a los valores extremos en el conjunto de datos aboga por un uso más frecuente, especialmente para la detección de valores atípicos.

#### Error estándar frente a desviación estándar

Tenemos una medida más de varianza para discutir: el error estándar de la media (SEM). El SEM a menudo se denomina simplemente error estándar (SE). Necesitamos volver a la población para entender qué es la SE y cuándo utilizarla. Si seleccionamos una muestra de la población, un conjunto de datos, podemos calcular la media de la muestra,  $\bar{x}$ . Si elegimos muestras repetidas y calculamos esas medias muestrales, generaremos un conjunto de datos de medias de las muestras de la población. Esto puede resultarle familiar; es el proceso que utilizamos para ilustrar el teorema del límite central en el capítulo 3. La desviación estándar del conjunto de medias es el error estándar.

La fórmula para el error estándar a partir de la desviación estándar es sencilla,

$$SE = \frac{s}{\sqrt{n}}$$

y no es más que una escala de la desviación estándar de la muestra por la raíz cuadrada del número de muestras.

¿Cuándo debemos usar la desviación estándar y cuándo debemos usar el error estándar? Utilice la desviación estándar para conocer la distribución de las muestras alrededor de la media.

Utilice el error estándar para decir algo sobre qué tan buena es una estimación de la media poblacional de una media muestral. En cierto sentido, el error estándar está relacionado tanto con el teorema del límite central, ya que afecta la desviación estándar de las medias de múltiples muestras de la población original, como con la ley de los números grandes, ya que es más probable que un conjunto de datos más grande dé una mejor estimación de la media poblacional.

Desde el punto de vista del aprendizaje profundo, podríamos utilizar la desviación estándar para describir el conjunto de datos utilizado para entrenar un modelo. Si entrenamos y probamos varios modelos, recordando la naturaleza estocástica de la inicialización profunda de la red, podemos calcular una media de los modelos para alguna métrica, digamos la precisión. En ese caso, es posible que deseemos informar la precisión media más o menos la

Error estándar. A medida que entrenamos más modelos y ganamos confianza en que la media La precisión representa el tipo de precisión que la arquitectura del modelo puede proporcionar. deberíamos esperar que el error en la precisión media de los modelos sea disminuir.

En resumen, en esta sección analizamos diferentes estadísticas resumidas, valores que podemos utilizar para comenzar a comprender un conjunto de datos. Estos incluyen los diversos medias (aritméticas, geométricas y armónicas), la mediana, la desviación estándar y, cuando corresponda, el error estándar. Por ahora, veamos cómo Podemos usar gráficos para ayudar a comprender un conjunto de datos.

## Cuantiles y diagramas de caja

Para calcular la mediana, necesitamos encontrar el valor medio, el número dividiendo el conjunto de datos en dos mitades. Matemáticamente, decimos que la mediana divide el conjunto de datos en dos cuantiles.

Un cuantil divide el conjunto de datos en grupos de tamaño fijo donde el tamaño fijo es el número de valores de datos en el cuantil. Dado que la mediana divide la conjunto de datos en dos grupos del mismo tamaño, es un 2 cuantil. A veces verás la mediana se conoce como percentil 50, es decir, el 50 por ciento de la los valores de los datos son menores que este valor. Entonces, mediante un razonamiento similar, el percentil 95 es el valor al que el 95 por ciento del conjunto de datos es inferior. Investigadores A menudo calculan 4 cuantiles y se refieren a ellos como cuartiles, ya que dividen los conjunto de datos en cuatro grupos de modo que el 25 por ciento de los valores de datos estén en el primer cuartil, el 50 por ciento está en el primero y segundo, y el 75 por ciento está en el primero, segundo y tercero, con el 25 por ciento final en el cuarto cuartil.

Analicemos un ejemplo para comprender lo que queremos decir con cuantiles. El ejemplo utiliza un conjunto de datos de examen sintético que representa 1000 pruebas. puntuaciones. Consulte el archivo exams.npy. Usaremos NumPy para calcular los valores de cuartil y luego trazaremos un histograma del conjunto de datos con los valores de cuartil. marcado. Primero, calculemos las posiciones de los cuartiles:

---

```
d = np.load("exámenes.npy")
p = d[:,0].astype("uint32")
q = np.quantile(p, [0,0, 0,25, 0,5, 0,75, 1,0])

imprimir("Cuartiles: ", q)
print("Cuenta por cuartil:")
print(" %d" % ((q[0] <= p) & (p < q[1])).sum())
print(" %d" % ((q[1] <= p) & (p < q[2])).sum())
print(" %d" % ((q[2] <= p) & (p < q[3])).sum())
print(" %d" % ((q[3] <= p) & (p < q[4])).sum())
```

---

Este código, junto con el código para generar el gráfico, se encuentra en el archivo quantiles.py.

Primero cargamos los datos del examen sintético y mantenemos las puntuaciones del primer examen (p). Tenga en cuenta que hacemos de p una matriz de números enteros para poder usar np.bincount más adelante para hacer el histograma. (Ese código no se muestra arriba). Luego usamos la función np.quantile de NumPy para calcular los valores del cuartil. Esta función toma la matriz fuente y una matriz de valores cuantiles en el rango [0, 1]. Los valores son fracciones de la distancia desde el valor mínimo de la matriz hasta su máximo. Entonces, pedir el cuantil 0,5 es pedir el valor que es la mitad de la distancia entre el mínimo de p y su máximo, de modo que el número de valores en cada conjunto sea igual.

Para obtener cuartiles, solicitamos los cuantiles 0,25, 0,5 y 0,75 para obtener valores tales que el 25 por ciento, el 50 por ciento y el 75 por ciento de los elementos de p sean menores que los valores. También solicitamos los cuantiles 0,0 y 1,0, el mínimo y el máximo de p. Hacemos esto por conveniencia cuando contamos el número de elementos en cada rango. Tenga en cuenta que, en su lugar, podríamos haber utilizado la función np.percentile . Devuelve los mismos valores que np.quantile pero utiliza valores porcentuales en lugar de fracciones. En ese caso, el segundo argumento habría sido [0,25,50,75,100].

Los valores de cuartil devueltos están en q. Los imprimimos para obtener

---

18,0, 56,75, 68,0, 78,0, 100,0

---

Aquí, 18 es el mínimo, 100 es el máximo y los tres valores límite para los cuartiles son 56,75, 68 y 78. Tenga en cuenta que el límite para el segundo cuartil es la mediana, 68.

El código restante cuenta el número de valores de p en cada rango. Con 1000 valores, esperaríamos tener 250 en cada rango, pero debido a que las matemáticas no siempre coinciden con los valores de datos existentes, obtenemos en su lugar

---

250, 237, 253, 248

---

lo que significa que 250 elementos de p son menores que 56,75, 237 están en [56,75, 68], y así adelante.

El código anterior utiliza un truco de conteo inteligente que vale la pena explicar. Queremos para contar el número de valores en p en algún rango. No podemos usar la función np.where de NumPy , ya que no le gusta la declaración condicional compuesta. Sin embargo, si usamos una expresión como  $10 \leq p$ , se nos dará una matriz del mismo tamaño que p donde cada elemento es Verdadero si la condición es verdadera para ese elemento o Falso si no lo es. Por lo tanto, al solicitar  $10 \leq p$  y  $p < 90$  se devolverán dos matrices booleanas. Para obtener los elementos en los que ambas condiciones son verdaderas, necesitamos unirlos lógicamente con un AND (&). Esto nos da una matriz final del mismo tamaño y forma que p, donde todos los elementos verdaderos representan valores en p en [10, 90). Para obtener el recuento, aplicamos el método de suma que, para una matriz booleana, trata a Verdadero como uno y Falso como cero.

La Figura 4-3 muestra el histograma de los datos del examen con los cuartiles marcado.

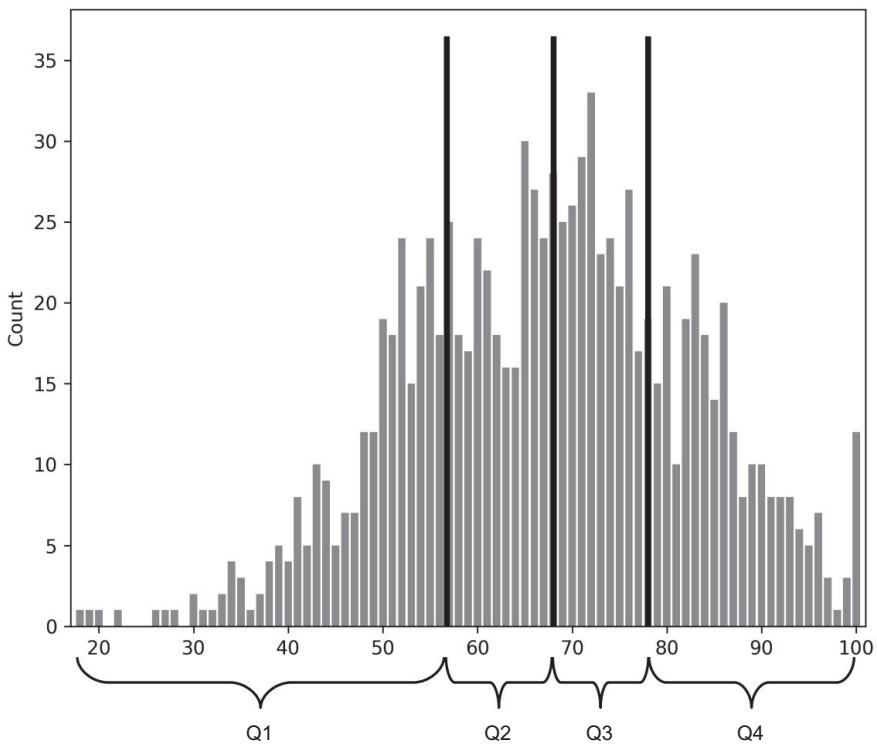


Figura 4-3: Histograma de 1000 calificaciones de exámenes con los cuartiles marcados

El ejemplo anterior muestra una vez más lo útil que es un histograma para visualizar y comprender datos. Deberíamos utilizar histogramas siempre que sea posible para ayudar a comprender qué sucede con un conjunto de datos. La figura 4-3 superpone los valores del cuartil en el histograma. Esto nos ayuda a entender lo que

son los cuartiles y su relación con los valores de los datos, pero esto no es un estilo típico de presentación. Más típico y útil porque puede mostrar múltiples características de un conjunto de datos es el diagrama de caja. Usémoslo ahora para las puntuaciones de los exámenes. arriba, pero esta vez también incluiremos los otros dos conjuntos de calificaciones de exámenes que ignorado anteriormente.

Primero mostraremos un diagrama de caja y luego lo explicaremos. Para ver el diagrama de caja de los tres exámenes en el archivo exams.npy, use

---

```
d = np.load("exámenes.npy")
plt.boxplot(d)
plt.xlabel("Prueba")
plt.ylabel("Puntuaciones")
plt.mostrar()
```

---

donde cargamos el conjunto completo de puntuajes de exámenes y luego usamos Matplotlib función de diagrama de caja .

Eche un vistazo al resultado, que se muestra en la Figura 4-4.

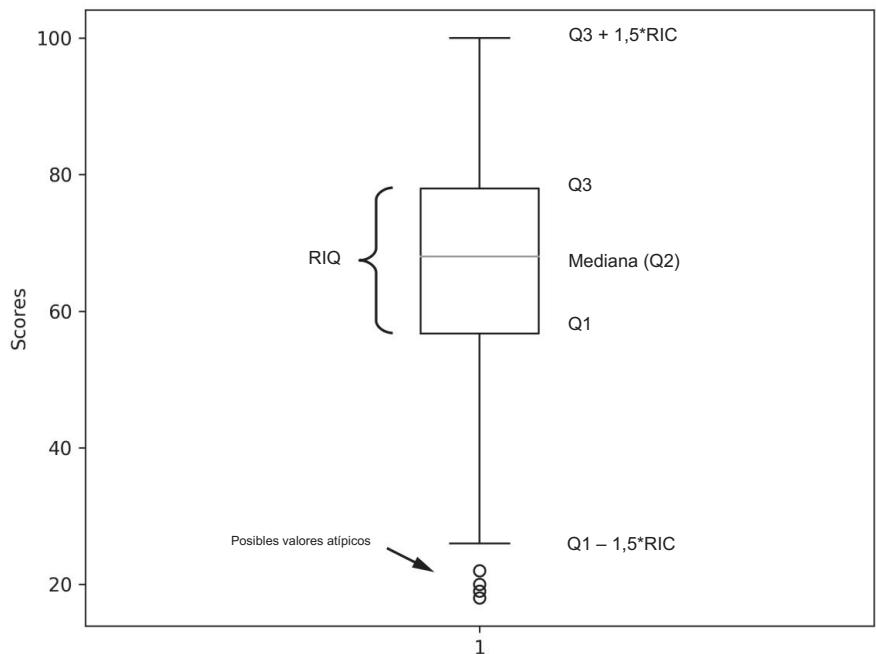
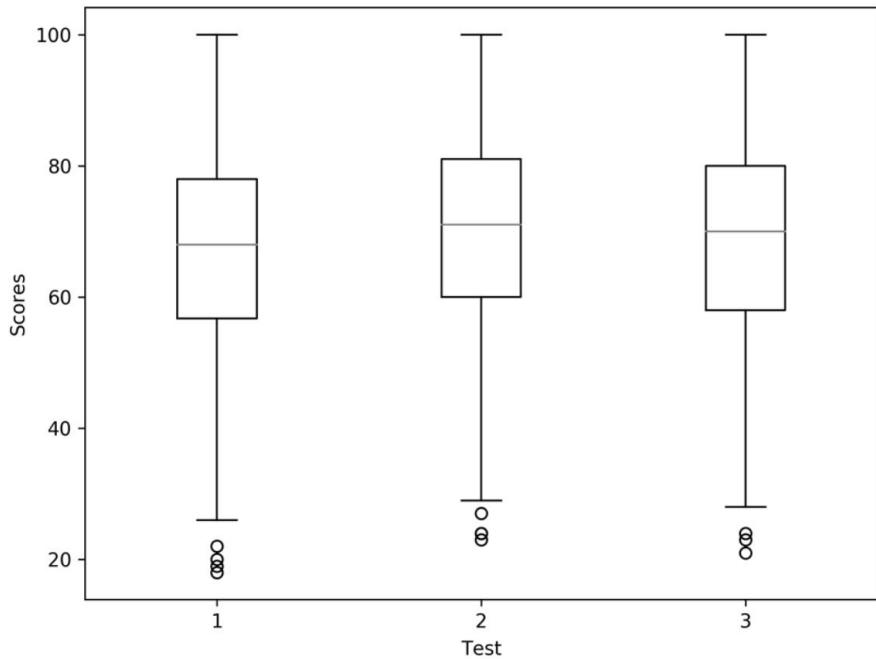


Figura 4-4: Diagramas de caja para los tres exámenes (arriba) y diagrama de caja para el primer examen con los componentes marcados (abajo)

El gráfico superior de la Figura 4-4 muestra el diagrama de caja de los tres conjuntos de puntuaciones de exámenes en `jexams.npy`. El primero de ellos se representa nuevamente en la parte inferior de la Figura 4-4, junto con etiquetas que describen las partes del gráfico.

Un diagrama de caja nos muestra un resumen visual de los datos. El cuadro del gráfico inferior de la Figura 4-4 ilustra el rango entre los límites para el primer cuartil (Q1) y el tercer cuartil (Q3). La diferencia numérica entre Q3 y Q1 se conoce como rango intercuartil (IQR). Cuanto mayor es el IQR, más dispersos están los datos alrededor de la mediana. Observe que esta vez la puntuación está en el eje y. Fácilmente podríamos haber hecho que el gráfico fuera horizontal, pero lo predeterminado es vertical. La mediana (Q2) está marcada cerca del centro del cuadro.

La media no se muestra en un diagrama de caja.

El diagrama de caja incluye dos líneas adicionales, los bigotes, aunque Matplotlib los llama volantes. Como se indicó, son 1,5 veces el RIQ por encima del Q3 o por debajo del Q1. Finalmente, hay algunos círculos etiquetados como "posibles valores atípicos". Por convención, los valores fuera de los bigotes se consideran posibles valores atípicos, lo que significa que podrían representar datos erróneos, ya sea ingresados incorrectamente a mano o, más probablemente hoy en día, recibidos de sensores defectuosos. Por ejemplo, los píxeles brillantes o muertos de una cámara CCD podrían considerarse valores atípicos. Al evaluar un conjunto de datos potencial, debemos ser sensibles a los valores atípicos y utilizar nuestro mejor criterio sobre qué hacer con ellos. Por lo general, solo hay unos pocos y podemos eliminar las muestras del conjunto de datos sin sufrir daños. Sin embargo, también es posible que los valores atípicos sean realmente reales y altamente indicativos de una clase particular. Si ese es el caso, queremos mantenerlos en el conjunto de datos con la esperanza de que el modelo los utilice de manera efectiva. La experiencia, la intuición y el sentido común deben guiarnos aquí.

Interpretemos el cuadro superior de la Figura 4-4 que muestra los tres conjuntos de puntuaciones de exámenes. La parte superior de los bigotes está en 100 cada vez, lo cual tiene sentido: un 100 es una puntuación perfecta y había cientos en el conjunto de datos. Observe que la parte del cuadro del gráfico no está centrada verticalmente en los bigotes. Si recordamos que el 50 por ciento de los valores de los datos están entre Q1 y Q3, con un 25 por ciento por encima y por debajo del Q2 en el cuadro, vemos que los datos no son rigurosamente normales; su distribución se desvía de una curva normal. Una mirada retrospectiva al histograma de la Figura 4-3 confirma esto para el primer examen. De manera similar, vemos que el segundo y tercer examen también se desvían de la normalidad. Entonces, un diagrama de caja puede decirnos qué tan similar es la distribución del conjunto de datos a una distribución normal. Cuando analicemos las pruebas de hipótesis a continuación, querremos saber si los datos se distribuyen normalmente o no.

¿Qué pasa con los posibles valores atípicos, los valores inferiores a  $Q1 - 1,5 \times IQR$ ? Sabemos que el conjunto de datos representa puntuaciones de exámenes, por lo que el sentido común nos dice que no se trata de valores atípicos, sino puntuaciones válidas de estudiantes particularmente confundidos (o perezosos). Si el conjunto de datos contuviera valores superiores a 100 o inferiores a cero, sería justo etiquetarlos como valores atípicos.

A veces, descartar muestras con valores atípicos es lo correcto. Sin embargo, si el valor atípico se debe a datos faltantes, cortar la muestra podría no ser una opción. Echemos un vistazo a lo que podríamos hacer con los datos faltantes y por qué, en general, deberíamos evitarlos como si fueran una plaga.

## Datos perdidos

Los datos que faltan son solo eso, datos que no tenemos. Si el conjunto de datos consta de muestras que representan vectores de características, los datos faltantes aparecen como una o más características en una muestra que no se midieron por algún motivo. A menudo, los datos faltantes están codificados de alguna manera. Si el valor solo es positivo, una característica faltante podría marcarse con  $-1$  o, históricamente,  $-999$ . Si la característica se nos proporciona como una cadena, es posible que la cadena esté vacía. Para valores de punto flotante, se puede utilizar un valor que no sea un número (NaN). NumPy nos facilita la búsqueda de NaN en una matriz usando np.isnan:

---

```
>>> a = np.arange(10, dtype="float64") >>> a[3] = np.nan
>>> np.isnan(a[3])
```

```
Verdadero
>>> a[3] == np.nan Falso
>>>
a[3] es np.nan Falso
```

---

Tenga en cuenta que la comparación directa con np.nan con `==` o `is` no funciona; solo funciona la prueba con `np.isnan`.

La detección de datos faltantes es específica del conjunto de datos. Suponiendo que nos hayamos convencido de que faltan datos, ¿cómo los manejamos?

Generemos un pequeño conjunto de datos con valores faltantes y usemos nuestro conocimiento estadístico existente para ver cómo manejarlos. El código para lo siguiente está en miss.py. Primero, generamos un conjunto de datos de 1000 muestras, cada una con cuatro características:

---

```
norte = 1000
np.random.seed(73939133) x =
np.zeros((N,4)) x[:,0] =
5*np.random.random(N) x[:,1] =
np.random.normal( 10,1,tamaño=N) x[:,2] =
3*np.random.beta(5,2,N) x[:,3] =
0.3*np.random.lognormal(tamaño=N)
```

---

El conjunto de datos está en `x`. Arreglamos la semilla del número aleatorio para obtener un resultado reproducible. La primera característica está distribuida uniformemente. El segundo tiene una distribución normal, mientras que el tercero sigue una distribución beta y el cuarto una distribución lognormal.

Por el momento, a `x` no le faltan valores. Agreguemos algunos haciendo NaN de elementos aleatorios:

---

```
i = np.random.randint(0,N, tamaño=int(0.05*N)) x[i,0] = np.nan i =
np.random.randint(0,N,
tamaño=int(0.05*N )) x[i,1] = np.nan i = np.random.randint(0,N,
tamaño=int(0.05*N))
```

```

xi[1,2] = np.nan
i = np.random.randint(0,N, tamaño=int(0.05*N))
xi[1,3] = np.nan

```

---

El conjunto de datos ahora tiene NaN en el 5 por ciento de sus valores.

Si a algunas muestras en un conjunto de datos grande les faltan datos, podemos eliminar del conjunto de datos sin preocupaciones. Sin embargo, si al 5 por ciento de las muestras le faltan datos, probablemente no queramos perder tantos datos. Más

Aún más preocupante, ¿qué pasa si hay una correlación entre los datos faltantes y una clase concreta? Desechar las muestras podría sesgar el conjunto de datos en algunos casos. manera que hará que el modelo sea menos útil.

¿Entonces, qué podemos hacer? Pasamos muchas páginas aprendiendo cómo resumir un conjunto de datos con estadísticas descriptivas básicas. ¿Podemos usarlos? Por supuesto. Podemos observar las distribuciones de las características, ignorando los valores faltantes, y usar esas distribuciones para decidir cómo queremos reemplazar las características. datos perdidos. Ingenuamente, usaríamos la media de los datos que tenemos, pero mirando en la distribución puede o no empujarnos hacia la mediana, dependiendo de qué tan lejos esté la distribución de lo normal. Esto suena como un trabajo para un diagrama de caja. Afortunadamente para nosotros, la función boxplot de Matplotlib es inteligente; ignora los NaN. Por lo tanto, hacer el diagrama de caja es una simple llamada a diagrama de caja (x).

La Figura 4-5 nos muestra el conjunto de datos con los NaN ignorados.

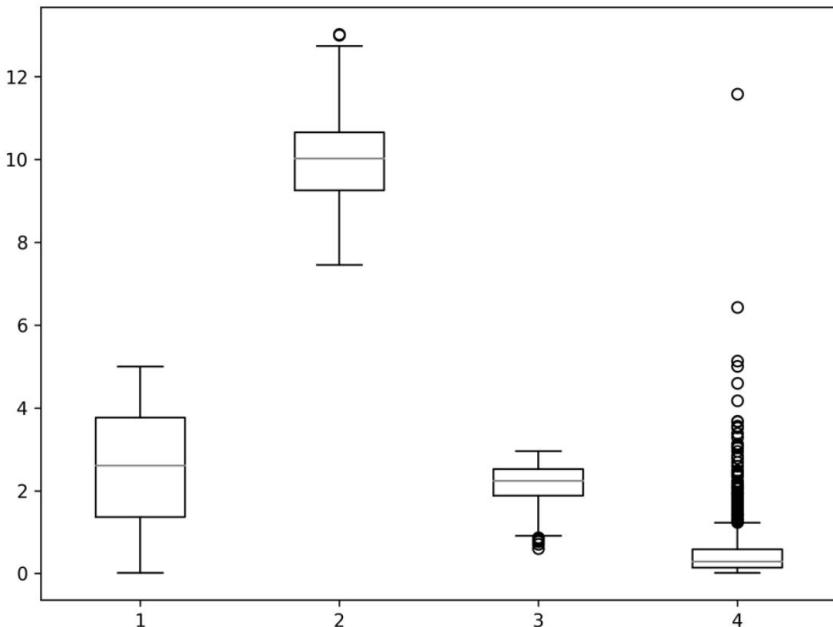


Figura 4-5: Diagrama de caja del conjunto de datos ignorando los valores faltantes

Los cuadros de la Figura 4-5 tienen sentido para las distribuciones de las características. La característica 1 está distribuida uniformemente, por lo que esperamos un cuadro simétrico alrededor de la media/mediana. (Estos son los mismos para la distribución uniforme). La característica 2 se distribuye normalmente, por lo que obtenemos una estructura de caja similar a la característica 1, pero, con solo 1000 muestras, es evidente cierta asimetría. La distribución beta de la Característica 3 está sesgada hacia la parte superior de su rango, como vemos en el diagrama de caja. Finalmente, la distribución lognormal de la Característica 4 debería estar sesgada hacia valores más bajos, con una larga cola visible como los muchos "valores atípicos" sobre los bigotes, una lección objetiva en contra de llamar irreflexivamente a esos valores como valores atípicos.

Debido a que tenemos características que no se distribuyen normalmente, actualice los valores faltantes con la mediana en lugar de la media. El código es sencillo:

---

```
good_idx = np.where(np.isnan(x[:,0]) == False) m =
np.median(x[good_idx,0]) bad_idx
= np.where(np.isnan(x[:,0]) == Verdadero) x[bad_idx,0]
= m
```

---

Aquí, primero tengo los índices de la Característica 1 que no son NaN. Los usamos para calcular la mediana (m). A continuación, asignamos i a los índices que son NaN y los reemplazamos con la mediana. Podemos hacer lo mismo con las otras funciones, actualizando todo el conjunto de datos para que ya no nos falten valores.

¿Hemos causado un gran cambio con respecto a las distribuciones anteriores? No, porque solo actualizamos el 5 por ciento de los valores. Por ejemplo, para la Característica 3, según la distribución beta, la media y las desviaciones estándar cambian así:

---

```
media no NaN, estándar = 2,169986, 0,474514
media actualizada, estándar = 2,173269, 0,462957
```

---

La moraleja de la historia es que si faltan suficientes datos como para que el conjunto de datos pueda quedar sesgado al eliminarlo, lo más seguro es reemplazar los datos faltantes con la media o mediana. Para decidir si utilizar la media o la mediana, consulte estadísticas descriptivas, un diagrama de caja o un histograma.

Además, si el conjunto de datos está etiquetado, como lo estaría un conjunto de datos de aprendizaje profundo, el proceso descrito anteriormente debe completarse con la media o mediana de muestras agrupadas por cada clase. De lo contrario, el valor calculado podría ser inadecuado para la clase.

Una vez eliminados los datos faltantes, se pueden entrenar modelos de aprendizaje profundo el conjunto de datos.

## Correlación

A veces, existe una asociación entre las características de un conjunto de datos. Si uno sube, el otro podría subir también, aunque no necesariamente de forma lineal. O bien, el otro podría desaparecer: una asociación negativa. El correcto

La palabra para este tipo de asociación es correlación. Una estadística que mide la correlación es una forma útil de comprender cómo se relacionan las características de un conjunto de datos.

Por ejemplo, no es difícil ver que los píxeles de la mayoría de las imágenes son altamente correlacionados. Esto significa que si seleccionamos un píxel al azar y luego un píxel adyacente, hay muchas posibilidades de que el segundo píxel sea similar al primero.

Las imágenes en las que esto no es cierto nos parecen ruido aleatorio.

En el aprendizaje automático tradicional, las características altamente correlacionadas no eran deseables, ya que no agregaban ninguna información nueva y solo servían para confundir los modelos. Todo el arte de la selección de funciones se desarrolló, en parte, para eliminar este efecto. Para el aprendizaje profundo moderno, donde la propia red aprende una nueva representación de los datos de entrada, es menos crítico tener entradas no correlacionadas. Esta es, en parte, la razón por la que las imágenes funcionan como entradas para redes profundas cuando normalmente no funcionan en absoluto con modelos de aprendizaje automático más antiguos.

Si el aprendizaje es tradicional o moderno, como parte del resumen. y explorar un conjunto de datos, vale la pena examinar y comprender las correlaciones entre las características. En esta sección, analizaremos dos tipos de correlaciones. Cada tipo devuelve un número único que mide la fuerza de la correlación entre dos características del conjunto de datos.

### correlación de Pearson

El coeficiente de correlación de Pearson devuelve un número,  $r \in [-1, +1]$ , que indica la fuerza de la correlación lineal entre dos características. Por lineal nos referimos a la fuerza con la que podemos describir la correlación entre las características mediante una línea. Si la correlación es tal que una característica aumenta exactamente cuando aumenta la otra característica, el coeficiente de correlación es  $+1$ . Por el contrario, si la segunda característica disminuye exactamente cuando la otra aumenta, la correlación es  $-1$ . Una correlación de cero significa que no hay asociación entre las dos características; son (posiblemente) independientes.

Se me pasó la palabra posiblemente en la oración anterior porque hay situaciones en las que una dependencia no lineal entre dos características podría conducir a un coeficiente de correlación de Pearson de cero. Sin embargo, estas situaciones no son comunes y, para nuestros propósitos, podemos afirmar que un coeficiente de correlación cercano a cero indica que las dos características son independientes. Cuanto más cercano a cero sea el coeficiente de correlación, ya sea positivo o negativo, más débil será la correlación entre las características.

La correlación de Pearson se define utilizando las medias de las dos características o las medias de los productos de las dos características. Las entradas son dos características, dos columnas del conjunto de datos. Llamaremos a estas entradas  $X$  e  $Y$ , donde la letra mayúscula se refiere a un vector de valores de datos. Tenga en cuenta que, dado que se trata de dos características del conjunto de datos,  $X_i$  está emparejado con  $Y_i$ , lo que significa que ambos provienen del mismo vector de características.

La fórmula para el coeficiente de correlación de Pearson es

$$\text{corr}(X, Y) = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E(X)^2} \sqrt{E(Y^2) - E(Y)^2}} \quad (4.5)$$

Hemos introducido una notación nueva, pero de uso común. La media de  $X$  es la expectativa de  $X$ , denotada como  $E(X)$ . Por lo tanto, en la ecuación 4.5, vemos la media de  $X$ ,  $E(X)$  y la media de  $Y$ ,  $E(Y)$ . Como podríamos sospechar,  $E(XY)$  es la media del producto de  $X$  e  $Y$ , elemento por elemento. De manera similar,  $E(X^2)$  es la media del producto de  $X$  consigo mismo, y  $E(X^2)$  es el cuadrado de la media de  $X$ . Con esta notación en la mano, podemos escribir fácilmente nuestra propia función para calcular la correlación de Pearson de dos vectores de características:

---

```
importar numpy como
np def pearson(x,y):
    exy = (x*y).mean() ex
    = x.mean() ey =
    y.mean() exx =
    (x*x).mean() ex2 =
    x.mean()**2 eyy =
    (y*y).mean() ey2 =
    y.mean()**2 return
    (exy - ex*ey)/(np.sqrt(exx-ex2)*np. raíz cuadrada (eyy-ey2))
```

---

La función de Pearson implementa directamente la Ecuación 4.5.

Configuremos un escenario en el que podamos usar Pearson y compararlo con lo que proporcionan NumPy y SciPy. El código que sigue, incluida la definición de Pearson anterior, se encuentra en el archivo correlación.py.

Primero, crearemos tres vectores correlacionados,  $x$ ,  $y$  y  $z$ . Imaginamos que estas son características de un conjunto de datos, de modo que  $x[0]$  está emparejado con  $y[0]$  y  $z[0]$ . El código que necesitamos es

---

```
np.semilla.aleatoria(8675309)
norte = 100
x = np.linspace(0,1,N) + (np.random.random(N)-0.5) y =
np.random.random(N)*xz =
-0.1*np.random.random(N)* X
```

---

Observe que nuevamente estamos arreglando la semilla pseudoaleatoria de NumPy para que la salida sea reproducible. La primera característica,  $x$ , es una línea ruidosa de cero a uno. El segundo,  $y$ , sigue a  $x$  pero también tiene ruido debido a la multiplicación por un valor aleatorio en  $[0, 1]$ . Finalmente,  $z$  está correlacionado negativamente con  $x$  debido al coeficiente  $-0.1$ .

El gráfico superior de la Figura 4-6 traza los tres valores de características secuencialmente para ver cómo se rastrean entre sí. El gráfico inferior muestra los tres como puntos emparejados, con un valor en el eje  $x$  y el otro en el eje  $y$ .

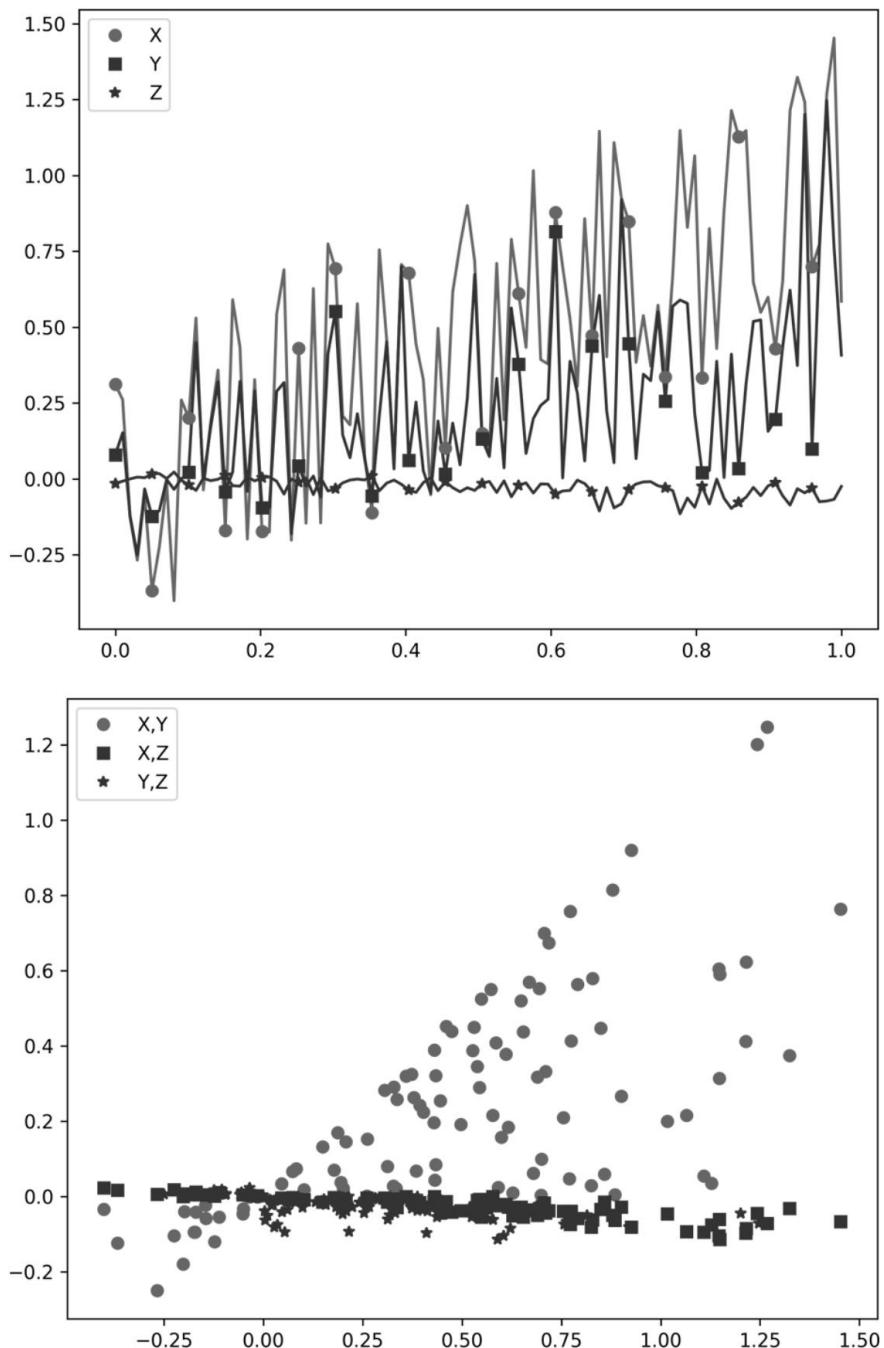


Figura 4-6: Tres características en secuencia para mostrar cómo se rastrean (arriba) y un diagrama de dispersión de las características como pares (abajo)

La función NumPy para calcular la correlación de Pearson es `np.corrcoef`.

A diferencia de nuestra versión, esta función devuelve una matriz que muestra las correlaciones

entre todos los pares de variables que se le pasan. Por ejemplo, usando nuestra función de Pearson , obtenemos lo siguiente como coeficientes de correlación entre x, y y z:

---

```
pearson(x,y): 0.682852
pearson(x,z): -0.850475
pearson(y,z): -0.565361
```

---

NumPy devuelve lo siguiente, con x, y y z apilados como una única matriz de  $3 \times 100$ :

---

```
>>> d = np.vstack((x,y,z))
>>> print(np.corrcoef(d))
[[ 1.  0.68285166 -0.85047468]
 [ 0.68285166 1. -0.56536104]
 [-0.85047468 -0.56536104 1. ]]
```

---

La diagonal corresponde a la correlación con cada característica y consigo misma, que es naturalmente perfecta y por tanto 1,0. La correlación entre xey está en el elemento 0,1 y coincide con el valor de nuestra función de Pearson . De manera similar, la correlación entre x y z está en el elemento 0,2, y la correlación entre y y z está en el elemento 1,2. Observe también que la matriz es simétrica, lo que esperamos porque  $\text{corr}(X, Y) = \text{corr}(Y, X)$ .

La función de correlación de SciPy es stats.pearsonr, que actúa como la nuestra pero devuelve un valor p junto con el valor r. Analizaremos más los valores p más adelante en el capítulo. Usamos el valor p devuelto como la probabilidad de que un sistema no correlacionado produzca el valor de correlación calculado. Para las características de nuestro ejemplo, el valor p es prácticamente idéntico a cero, lo que implica que no existe una probabilidad razonable de que un sistema no correlacionado haya producido las características.

Dijimos anteriormente que, en el caso de las imágenes, los píxeles cercanos suelen estar altamente correlacionados. Veamos si esto es realmente cierto para una imagen de muestra. Usaremos la imagen de China incluida con sklearn y trataremos filas específicas de la banda verde como vectores emparejados. Calcularemos el coeficiente de correlación para dos filas adyacentes, una fila más alejada y un vector aleatorio:

---

```
>>> from sklearn.datasets import load_sample_image
>>> china = load_sample_image('china.jpg')
>>> a = china[230,:,:1].astype("float64")
>>> b = china[231,:,:1].astype("float64")
>>> c = china[400,:,:1].astype("float64")
>>> d = np.random.random(640)
>>> pearson(a,b)
0.8979360
>>> pearson(a,c)
-0.276082
>>> pearson(a,d)
-0.038199
```

---

La comparación de las filas 230 y 231 muestra que tienen una correlación muy positiva. La comparación de las filas 230 y 400 muestra una correlación más débil y, en este caso, negativa. Finalmente, como era de esperar, la correlación con un vector aleatorio da un valor cercano a cero.

El coeficiente de correlación de Pearson se usa tan ampliamente que a menudo Considérelo referido simplemente como el coeficiente de correlación. Ahora echemos un vistazo a una segunda función de correlación y vea en qué se diferencia del coeficiente de Pearson.

### Correlación de lancero

La segunda medida de correlación que exploraremos es el coeficiente de correlación de Spearman,  $[-1, +1]$ . Es una medida basada en las clasificaciones de los valores de las características. en lugar de los valores mismos.

Para clasificar X, reemplazamos cada valor en X con el índice de ese valor en el versión ordenada de X. Si X es

---

[86, 62, 28, 43, 3, 92, 38, 87, 74, 11]

---

entonces los rangos son

---



---

[7, 5, 2, 4, 0, 9, 3, 8, 6, 1]

---

porque cuando se ordena X, 86 va en el octavo lugar (contando desde cero), y 3 va primero.

La correlación de Pearson busca una relación lineal, mientras que la Spearman busca cualquier asociación monótona entre las entradas.

Si tenemos los rangos para los valores de las características, entonces el coeficiente de Spearman es

$$= 1 - \left( \frac{6}{\frac{n(n-1)}{2} - 1) } \right) \sum_{i=0}^{n-1} d_i^2 \quad (4.6)$$

donde n es el número de muestras y  $d = \text{rango}(X) - \text{rango}(Y)$  es la diferencia del rango de los valores X e Y emparejados. Observe cómo la Ecuación 4.6 es sólo válido si las clasificaciones son únicas (es decir, no hay valores repetidos en X o Y).

Para calcular d en la ecuación 4.6, necesitamos clasificar X e Y y usar la diferencia de rangos. La correlación de Spearman es la correlación de Pearson de las categorías.

El ejemplo anterior señala el camino hacia una implementación de la correlación Spearman:

---

```
importar numpy como np
def lancero(x,y):
    n = largo(x)
    t = x[np.argsort(x)]
    rx = []
    para i en el rango (n):
        rx.append(t[i])
```

---

```

rx.append(np.donde(x[i] == t)[0][0])
rx = np.array(rx, dtype="float64") t =
y[np.argsort(y)] ry = []
para i
en el rango(n):
    ry.append(np.where(y[i] == t)[0][0])
ry = np.array(ry, dtype="float64") d = rx
- ry return
1.0 - (6.0/(n*(n*n-1)))*(d**2).sum()

```

---

Para obtener los rangos, primero debemos ordenar X (t). Luego, para cada valor en X (x), encontramos dónde ocurre en t vía np.where y tomamos el primer elemento, la primera coincidencia. Después de construir la lista rx , la convertimos en una matriz NumPy de punto flotante. Hacemos lo mismo para que Y se enrífe . Con los rangos, d se establece en su diferencia y se utiliza la ecuación 4.6 para devolver el valor de Spearman.

Tenga en cuenta que esta versión de la correlación de Spearman está limitada por la Ecuación 4.6 y debe usarse cuando no hay valores duplicados en X o Y. Nuestro ejemplo en esta sección usa valores de punto flotante aleatorios, por lo que la probabilidad de un duplicado exacto es bastante bajo.

Compararemos nuestra implementación de Spearman con la versión SciPy, stats.spearmanr. Al igual que la versión SciPy de la correlación de Pearson, stats.spearmanr devuelve un valor p. Lo ignoraremos. Veamos cómo se compara nuestra función:

---

```

>>> de scipy.stats importar lancero >>>
print(lanzador(x,y), lancero(x,y)[0])
0.694017401740174 0.6940174017401739
>>> print(lanzador(x,z), lancero(x,z )[0])
-0,8950855085508551 -0,895085508550855
>>> print(lancero(y,z), lancero(y,z)[0])
-0.6414041404140414 -0.6414041404140414

```

---

Estamos completamente de acuerdo con la función SciPy hasta el último bit del valor de punto flotante.

Es importante recordar la diferencia fundamental entre las correlaciones de Pearson y Spearman. Por ejemplo, considere la correlación entre una rampa lineal y la función sigmoidea:

---

```

rampa = np.linspace(-20,20,1000)
sig = 1.0 / (1.0 + np.exp(-ramp))
print(pearson(ramp,sig))
print(lancero(ramp,sig))

```

---

Aquí, la rampa aumenta linealmente de -20 a 20 y sig sigue una forma sigmoidea (curva "S"). La correlación de Pearson será alta, ya que ambas aumentan a medida que x se vuelve más positiva, pero la asociación no es puramente lineal. Ejecutar el ejemplo da

---

```

0.905328
1.0

```

---

indicando una correlación de Pearson de 0,9 pero una correlación de Spearman perfecta de 1,0, ya que por cada aumento de rampa hay un aumento de sig y solo un aumento. La correlación de Spearman ha capturado la relación no lineal entre los argumentos, mientras que la correlación de Pearson sólo la ha insinuado. Si analizamos un conjunto de datos destinado a un algoritmo clásico de aprendizaje automático, la correlación de Spearman podría ayudarnos a decidir qué características conservar y cuáles descartar.

Con esto concluye nuestro examen de las estadísticas para describir y comprender datos. Aprendamos ahora a utilizar las pruebas de hipótesis para interpretar resultados experimentales y responder preguntas como "¿Estos dos conjuntos de muestras de datos pertenecen a la misma distribución principal?"

## Evaluación de la hipótesis

Contamos con dos grupos independientes de 50 estudiantes que estudian biología celular. No tenemos motivos para creer que los grupos difieran de manera significativa, ya que los estudiantes de la población más grande fueron asignados al azar. El grupo 1 asistió a las conferencias y, además, trabajó mediante un conjunto estructurado de ejercicios por ordenador. El grupo 2 sólo asistió a las conferencias. Ambos grupos tomaron el mismo examen final, lo que condujo a las puntuaciones de las pruebas que se muestran en la Tabla 4-1. Queremos saber si pedir a los estudiantes que resolvieran los ejercicios de computadora marcó una diferencia en los puntajes finales de sus exámenes.

Tabla 4-1: Puntajes de las pruebas del Grupo 1 y del Grupo 2

---

Grupo 1	81	80	85	87	83	87	87	90	79	83	88	75	87	92	78	80	83	91	82
	88	89	92	97	82	79	82	82	85	89	91	83	85	77	81	90	87	82	
	84	86	79	84	85	90	84	90	85	85	78	94	100						

---

Grupo 2	92	82	78	74	86	69	83	67	85	82	81	91	79	82	82	88	80	63	85
	86	77	94	85	75	77	89	86	71	82	82	80	88	72	91	90	92	95	
	87	71	83	94	90	78	60	76	88	91	83	85	73						

---

La Figura 4-7 muestra un diagrama de caja de la Tabla 4-1.

Para entender si hay un cambio significativo en los puntajes de las pruebas finales entre los dos grupos, necesitamos probar algunas hipótesis. El método que usaremos para probar las hipótesis se conoce como prueba de hipótesis y es una pieza fundamental de la ciencia moderna.

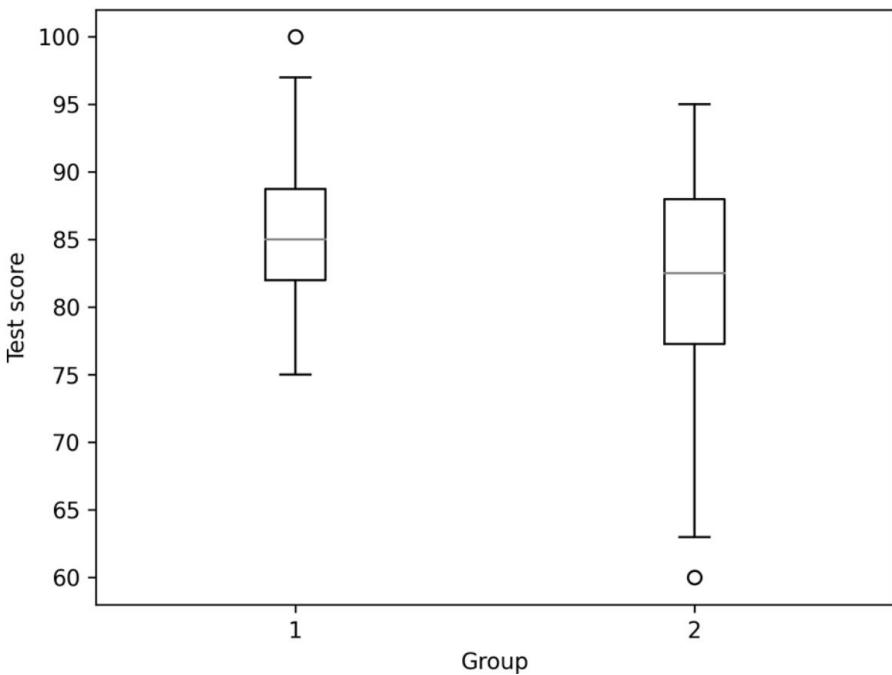


Figura 4-7: Diagrama de caja para los datos de la Tabla 4-1

La prueba de hipótesis es un tema amplio, demasiado extenso para que podamos ofrecer aquí más que una mínima introducción. Como este es un libro sobre aprendizaje profundo, nos centraremos en el escenario que probablemente encontrará un investigador de aprendizaje profundo. Consideraremos sólo dos pruebas de hipótesis: la prueba t para muestras no apareadas de varianza diferente (una prueba paramétrica) y la U de Mann-Whitney (una prueba no paramétrica). A medida que avancemos, entenderemos qué son estas pruebas y por qué nos restringimos a ellas, así como el significado de paramétrico y no paramétrico.

Para tener éxito con las pruebas de hipótesis, necesitamos saber qué entendemos por hipótesis, por lo que abordaremos eso primero, junto con nuestra justificación para limitar los tipos de pruebas de hipótesis que consideraremos. Con el concepto de hipótesis en la mano, analizaremos la prueba t y la prueba U de Mann-Whitney, utilizando los datos de la tabla 4-1 como ejemplo. Empecemos.

### Hipótesis

Para comprender si dos conjuntos de datos provienen de la misma distribución principal o no, podríamos observar estadísticas resumidas. La Figura 4-7 nos muestra el diagrama de caja para el Grupo 1 y el Grupo 2. Parece que los dos grupos tienen medias y desviaciones estándar diferentes. ¿Cómo sabemos? El diagrama de caja nos muestra

la ubicación de las medianas y los bigotes nos dicen algo sobre la varianza. Ambos juntos sugieren que las medias serán diferentes porque las medianas son diferentes y ambos conjuntos de datos son razonablemente simétricos alrededor de la mediana. El espacio entre los bigotes indica la desviación estándar. Entonces, hagamos hipótesis utilizando los medios de los conjuntos de datos.

En la prueba de hipótesis, tenemos dos hipótesis. La primera, conocida como hipótesis nula ( $H_0$ ), es que los dos conjuntos de datos provienen de la misma distribución principal y que no hay nada especial que los diferencie. La segunda hipótesis, la hipótesis alternativa ( $H_a$ ), es que los dos grupos no pertenecen a la misma distribución. Como usaremos las medias,  $H_0$  dice que las medias, en realidad las medias de la población principal que generó los datos, son las mismas. De manera similar, si rechazamos  $H_0$ , implícitamente estamos aceptando  $H_a$  y afirmando que tenemos evidencia de que las medias son diferentes. No tenemos las medias poblacionales reales, por lo que usaremos las medias muestrales y las desviaciones estándar.

La prueba de hipótesis no nos dice definitivamente si  $H_0$  es cierta. En cambio, nos da evidencia a favor de rechazar o aceptar la hipótesis nula. Es fundamental recordar esto.

Estamos probando dos muestras independientes para ver si deberíamos pensar en ellas, como provenientes de la misma distribución principal. Hay otras formas de utilizar las pruebas de hipótesis, pero rara vez las encontramos en el aprendizaje profundo. Para la tarea que nos ocupa, necesitamos las medias muestrales y las desviaciones estándar muestrales.

Nuestras pruebas plantearán la pregunta: "¿Existe una diferencia significativa en las medias de estos dos conjuntos?"

Sólo nos interesa detectar si los dos grupos de datos provienen de la misma distribución principal, por lo que otra simplificación que haremos es que todas nuestras pruebas serán bilaterales o bilaterales. Cuando usamos una prueba, como la prueba t que describiremos a continuación, comparamos nuestro estadístico de prueba calculado (el valor t) con la distribución del estadístico de prueba y hacemos preguntas sobre la probabilidad de que nuestro valor t calculado sea . Si queremos saber si el estadístico de prueba está por encima o por debajo de alguna fracción de esa distribución, estamos haciendo una prueba de dos colas. Si en cambio queremos saber acerca de la probabilidad de que el estadístico de prueba esté por encima de un valor particular sin preocuparnos de que esté por debajo, o viceversa, entonces estamos haciendo una prueba unilateral.

Expongamos nuestras suposiciones y enfoque:

1. Tenemos dos conjuntos de datos independientes que deseamos comparar.
2. No estamos asumiendo si las desviaciones estándar de los datos son las mismas.
3. Nuestra hipótesis nula es que las medias de las distribuciones principales de los conjuntos de datos son las mismas,  $H_0: \mu_1 = \mu_2$ . Usaremos las medias muestrales ( $\bar{x}_1$ ,  $\bar{x}_2$ ) y las desviaciones estándar muestrales ( $s_1$ ,  $s_2$ ) para ayudarnos a decidir aceptar o rechazar  $H_0$ .
4. Las pruebas de hipótesis suponen que los datos son independientes y están distribuidos idénticamente (iid). Interpretamos esto como una afirmación de que los datos son una muestra aleatoria justa.

Una vez entendidos estos supuestos, comenzemos con la prueba t, la prueba de hipótesis más utilizada.

### la prueba t

La prueba t depende de t, el estadístico de prueba. Esta estadística se compara con la distribución t y se usa para generar un valor p, una probabilidad que usaremos para llegar a una conclusión sobre H<sub>0</sub>. Hay una rica historia detrás de la prueba t y la prueba z relacionada que ignoraremos aquí. Le animo a que profundice en las pruebas de hipótesis cuando tenga la oportunidad o, como mínimo, revise artículos reflexivos sobre la forma correcta de realizar una prueba de hipótesis e interpretar sus resultados.

La prueba t es una prueba paramétrica. Esto significa que existen suposiciones sobre los datos y la distribución de los mismos. Específicamente, la prueba t supone, más allá de que los datos sean iid, que la distribución (histograma) de los datos es normal. Hemos dicho antes que muchos procesos físicos parecen seguir una distribución normal, por lo que hay motivos para pensar que los datos de mediciones reales podrían seguir esa misma distribución.

Hay muchas formas de probar si un conjunto de datos tiene una distribución normal, pero las ignoraremos, ya que existe cierto debate sobre la utilidad de dichas pruebas. En su lugar, le sugeriré (un poco imprudentemente) que utilice la prueba t y la prueba U de Mann-Whitney juntas para ayudarle a tomar una decisión sobre aceptar o rechazar H<sub>0</sub>.

El uso de ambas pruebas puede llevar a una situación en la que no estén de acuerdo, donde una prueba dice que hay evidencia en contra de la hipótesis nula y la otra dice que no la hay. En general, si la prueba no paramétrica pretende evidencia contra H<sub>0</sub>, entonces probablemente se debería aceptar esa evidencia independientemente del resultado de la prueba t.

Si el resultado de la prueba t coincide con H<sub>0</sub>, pero la prueba U de Mann-Whitney no lo es, y cree que los datos son normales, entonces también puede aceptar el resultado de la prueba t.

La prueba t tiene diferentes versiones. Hemos indicado explícitamente anteriormente que usaremos una versión diseñada para conjuntos de datos de diferentes tamaños y variaciones. La versión específica de la prueba t que usaremos es la prueba t de Welch, que no supone que la varianza de los dos conjuntos de datos sea la misma.

La puntuación t para la prueba t de Welch es

$$t = \frac{x_1 - x_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

donde n<sub>1</sub> y n<sub>2</sub> son el tamaño de los dos grupos.

La puntuación t y un valor asociado conocido como grados de libertad, que es similar pero también diferente de los grados de libertad mencionados anteriormente, genera la curva de distribución t adecuada. Para obtener un valor p, calculamos el área bajo la curva, tanto arriba como abajo (puntaje t positivo y negativo), y lo devolvemos. Dado que la integral de una distribución de probabilidad es 1, el área total debajo de las colas desde el valor de puntuación t positivo y negativo hasta el infinito positivo y negativo será el valor p. Usaremos los grados de libertad siguientes para ayudarnos a calcular los intervalos de confianza.

¿Qué nos dice el valor p? Nos dice la probabilidad de ver la diferencia entre las dos medias que vemos, o mayor, si se cumple la hipótesis nula.

es verdad. Normalmente, si esta probabilidad está por debajo de algún umbral que hemos elegido, rechazamos la hipótesis nula y decimos que tenemos evidencia de que los dos grupos tienen medias diferentes, es decir, que provienen de distribuciones parentales diferentes. Cuando rechazamos  $H_0$ , decimos que la diferencia es estadísticamente significativa. El umbral para aceptar/rechazar  $H_0$  se llama, generalmente con = 0,05 como valor típico, aunque problemático. Discutiremos por qué 0.05 es problemático a continuación.

El punto a recordar es que el valor  $p$  supone que la hipótesis nula es verdadera. Nos dice la probabilidad de que una  $H_0$  verdadera nos dé al menos la diferencia que vemos, o mayor, entre los grupos. Si el valor  $p$  es pequeño, eso tiene dos posibles significados: (1) la hipótesis nula es falsa, o (2) un error de muestreo aleatorio nos ha dado muestras que quedan fuera de lo que podríamos esperar. Dado que el valor  $p$  supone que  $H_0$  es verdadera, un valor  $p$  pequeño nos ayuda a creer cada vez menos en (2) y aumenta nuestra confianza en que (1) podría ser correcto. Sin embargo, el valor  $p$  por sí solo no puede confirmar (1); Es necesario que entren en juego otros conocimientos.

Mencioné que usar = 0,05 es problemático. La razón principal es El problema es que es demasiado generoso; conduce a demasiados rechazos de una hipótesis nula verdadera. Según James Berger y Thomas Sellke en su artículo "Testing a Point Null Hypothesis: The Irreconcilability of P Values and Evidence" (Revista de la Asociación Estadounidense de Estadística, 1987), cuando

= 0,05, se rechazará alrededor del 30 por ciento de las hipótesis nulas verdaderas. Cuando usamos algo como  $\leq 0,001$ , la probabilidad de rechazar falsamente una hipótesis nula verdadera se reduce a menos del 3 por ciento. La moraleja de la historia es que  $p < 0,05$  no es mágico y, francamente, no es convincente para un solo estudio. Busque valores  $p$  altamente significativos de al menos 0,001 o, preferiblemente, mucho más pequeños. En  $p = 0,05$ , todo lo que tienes es una sugerencia y debes repetir el experimento. Si todos los experimentos repetidos tienen un valor  $p$  de alrededor de 0,05, entonces rechazar la hipótesis nula comienza a tener sentido.

#### Intervalos de confianza

Junto con un valor  $p$ , a menudo verá intervalos de confianza (IC). El intervalo de confianza proporciona límites dentro de los cuales creemos que se ubicará la verdadera diferencia poblacional en las medias, con una confianza determinada para muestras repetidas de los dos conjuntos de datos que estamos comparando. Normalmente, informamos intervalos de confianza del 95 por ciento. Nuestras pruebas de hipótesis verifican la igualdad de medias preguntando si la diferencia de las medias muestrales es cero o no. Por lo tanto, cualquier IC que incluya cero nos indica que no podemos rechazar la hipótesis nula.

Para la prueba  $t$  de Welch, los grados de libertad son

$$df = \frac{\left( \frac{s_1^2}{n_1} + \frac{s_2^2}{n_2} \right)^2}{\frac{(\bar{s}_1/n_1)^2}{n_1-1} \left( s + \frac{\bar{s}_2/n_2}{n_2-1} \right)^2} \quad (4.7)$$

que podemos utilizar para calcular intervalos de confianza,

$$CI = \left( \bar{x}_1 - \bar{x}_2 \right) \pm t_{1-\alpha/2, df} \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \quad (4.8)$$

donde  $t_{1-\alpha/2, df}$  es el valor crítico, y el valor  $t$  para el nivel de confianza dado ( $\alpha$ ) y los grados de libertad,  $df$ , provienen de la Ecuación 4.7.

¿Cómo debemos interpretar el intervalo de confianza del 95 por ciento? Hay un valor poblacional: la verdadera diferencia entre las medias del grupo. El intervalo de confianza del 95 por ciento es tal que si pudiéramos extraer muestras repetidas de la distribución que produjo los dos conjuntos de datos, el 95 por ciento de los intervalos de confianza calculados contendría la verdadera diferencia entre las medias. No es el rango que incluye la verdadera diferencia en las medias con una certeza del 95 por ciento.

Más allá de verificar si hay cero en el IC, el IC es útil porque su ancho nos dice algo sobre la magnitud del efecto. Aquí, el efecto está relacionado con la diferencia entre las medias. Es posible que tengamos una diferencia estadísticamente significativa según el valor  $p$ , pero el efecto podría carecer prácticamente de significado. El IC será estrecho cuando el efecto sea grande porque los IC pequeños implican un rango estrecho que abarca el efecto real. En breve veremos cómo, cuando sea posible, calcular otra medida útil del efecto.

Finalmente, un valor  $p$  menor que también tendrá un IC que no incluye  $H_0$ .

En otras palabras, lo que nos dice el valor  $p$  y lo que nos dice el intervalo de confianza no se contradicen entre sí.

### Tamaño

del efecto Una cosa es tener un valor  $p$  estadísticamente significativo. Otra es que la diferencia representada por ese valor  $p$  sea significativa en el mundo real.

Una medida popular del tamaño de un efecto, el tamaño del efecto, es la  $d$  de Cohen. Para nosotros, dado que utilizamos la prueba  $t$  de Welch, la  $d$  de Cohen se encuentra calculando

$$d = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \quad (4.9)$$

La  $d$  de Cohen suele interpretarse subjetivamente, aunque debemos informar el valor numérico también. Subjetivamente, el tamaño del efecto podría ser

$d$	Efecto
0,2	Pequeño
0,5	Mediano
0,8	Grande

La  $d$  de Cohen tiene sentido. La diferencia entre las medias es una forma natural de pensar en el efecto. Escalarlo según la varianza media lo coloca en un rango consistente. De la ecuación 4.9 vemos que un valor  $p$  correspondiente a un resultado estadísticamente significativo podría conducir a un efecto pequeño que no tiene ninguna importancia práctica real.

#### Evaluación de los puntajes

de las pruebas Juntemos todo lo anterior para aplicar la prueba t a los datos de nuestra prueba de la Tabla 4-1. Encontrarás el código en el archivo hipótesis.py. Primero generamos los conjuntos de datos:

---

```
np.random.seed(65535) a =
np.random.normal(85,6,50).astype("int32") a[np.where(a > 100)]
= 100 b =
np.random.normal( 82,7,50).astype("int32") b[np.where(b >
100)] = 100
```

---

Una vez más, estamos utilizando una semilla numérica pseudoaleatoria fija de NumPy para lograr repetibilidad. Hacemos una muestra a partir de una distribución normal con una media de 85 y una desviación estándar de 6,0. Seleccionamos  $b$  de una distribución normal,

con una media de 82 y una desviación estándar de 7,0. Para ambos, limitamos cualquier valor por encima de 100 a 100. Después de todo, se trata de puntuaciones de exámenes, sin crédito adicional.

Aplicamos la prueba t a continuación:

---

```
de scipy.stats importar ttest_ind t,p =
ttest_ind(a,b, equal_var=False) print("(t=%0.5f,
p=%0.5f) % (t,p))
```

---

Obtenemos ( $t = 2.40234$ ,  $p = 0.01852$ ). La  $t$  es la estadística y  $p$  es el valor  $p$  calculado. Es 0,019, que es menor que 0,05 pero sólo por un factor de dos.

Tenemos un resultado débil que nos dice que podríamos querer rechazar la hipótesis nula y creer que los dos grupos,  $a$  y  $b$ , provienen de distribuciones diferentes.

Por supuesto, sabemos que sí porque los generamos, pero es bueno ver que la prueba apunta en la dirección correcta.

Observe que la función que importamos de SciPy es `ttest_ind`. Esta es la función a utilizar para muestras independientes, que no están emparejadas. Además, observe que agregamos `igualdad_var=False` a la llamada. Así es como se utiliza la prueba t de Welch, que no supone que la varianza entre los dos conjuntos de datos sea igual.

Sabemos que no son iguales, ya que  $a$  usa una desviación estándar de 6,0 mientras que  $b$  usa 7,0.

Para obtener los intervalos de confianza, escribiremos una función `CI`, ya que NumPy y SciPy no incluyen ninguna. La función implementa directamente las Ecuaciones 4.7 y 4.8:

---

```
de las estadísticas de importación
de scipy def CI (a, b, alfa = 0,05):
    n1, n2 = longitud(a), longitud(b)
```

---

---

```
s1, s2 = np.std(a, ddof=1)**2, np.std(b, ddof=1)**2 df
= (s1/n1 + s2/n2)**2 / ((s1/n1)**2/(n1-1) + (s2/n2)**2/(n2-1)) tc =
stats.t.ppf(1 - alfa/2, df) lo =
(a.media() - b.media()) - tc*np.sqrt(s1/n1 + s2/n2) hola
= (a.media() - b.media()) + tc*np.sqrt(s1/n1 + s2/n2)
volver lo, hola
```

---

El valor t crítico se obtiene llamando a stats.t.ppf, pasando el valor /2 y los grados de libertad adecuados, df. El valor t crítico es el valor percentil del 97,5 por ciento, para = 0,05, que es lo que devuelve la función de punto porcentual (ppf). Dividimos por dos para cubrir las colas de la distribución t.

Para nuestro ejemplo de prueba, el intervalo de confianza es [0,56105, 5,95895]. Note observe cómo esto no incluye cero, por lo que el IC también indica un resultado estadísticamente significativo. Sin embargo, el rango es bastante grande, por lo que este no es un resultado particularmente sólido. El rango de IC puede ser difícil de interpretar por sí solo, así que, finalmente, calculemos la d de Cohen para ver si tiene sentido dada la amplitud del intervalo de confianza. En código, implementamos la Ecuación 4.9:

---

```
def Cohen_d(a,b):
    s1 = np.std(a, ddof=1)**2
    s2 = np.std(b, ddof=1)**2
    return (a.media() - b.media()) / np.sqrt(0.5*(s1+s2))
```

---

Obtenemos d = 0,48047, correspondiente a un tamaño del efecto medio.

### La prueba U de Mann-Whitney

La prueba t supone que la distribución de los datos fuente es normal. Si los datos no se distribuyen normalmente, deberíamos utilizar una prueba no paramétrica. Las pruebas no paramétricas no hacen suposiciones sobre la distribución subyacente de los datos. La prueba U de Mann-Whitney, a veces llamada prueba de suma de rangos de Wilcoxon, es una prueba no paramétrica que ayuda a decidir si dos conjuntos diferentes de datos provienen de la misma distribución principal. La prueba U de Mann-Whitney no se basa directamente en los valores de los datos, sino que utiliza la clasificación de los datos.

La hipótesis nula para esta prueba es la siguiente: la probabilidad de que un valor seleccionado al azar del Grupo 1 sea mayor que un valor seleccionado al azar del Grupo 2 es 0,5. Pensemos un poco en eso. Si los datos provienen de la misma distribución principal, entonces deberíamos esperar que cualquier par de valores seleccionados aleatoriamente de los dos grupos no muestre preferencia sobre cuál es mayor que el otro.

La hipótesis alternativa es que la probabilidad de que un valor seleccionado al azar del Grupo 1 sea mayor que un valor seleccionado al azar del Grupo 2 no es 0,5. Observe que no hay ninguna afirmación sobre si la probabilidad es mayor o menor que 0,5, sólo que no es 0,5; por tanto, la prueba U de Mann-Whitney, tal como la usaremos, es bilateral.

La hipótesis nula de la prueba U de Mann-Whitney no es la misma que la hipótesis nula de la prueba t. Para la prueba t, preguntamos si las medias

entre los dos grupos son iguales. (En realidad, estamos preguntando si la diferencia en las medias es cero). Sin embargo, si dos conjuntos de datos provienen de distribuciones principales diferentes, ambas hipótesis nulas son falsas, por lo que podemos usar la prueba U de Mann-Whitney en su lugar. de la prueba t, especialmente cuando los datos subyacentes no se distribuyen normalmente.

Para generar U, el estadístico de Mann-Whitney, primero agrupamos ambos conjuntos de datos y los clasificamos. Los empates se reemplazan con la media entre el rango del valor del empate y el siguiente valor del rango. También realizamos un seguimiento del grupo de origen para poder separar la lista de rangos nuevamente. Las clasificaciones, por grupo, se suman para dar R1 y R2 (utilizando las clasificaciones de los datos agrupados). Calculamos dos valores,

$$U_1 = n_1 n_2 + \frac{n_1 (n_1 - 1)}{2} - R_1$$

$$U_2 = n_1 n_2 + \frac{n_2 (n_2 - 1)}{2} - R_2$$

con el más pequeño llamado U, el estadístico de prueba. Es posible generar un valor p a partir de U, teniendo en cuenta toda la discusión anterior sobre el significado y el uso de los valores p. Como antes, n1 y n2 son el número de muestras en los dos grupos. La prueba U de Mann-Whitney requiere que el menor de estos dos números sea al menos 21 muestras. Si no tiene tantos, es posible que los resultados no sean confiables al utilizar la función SciPy mannwhitneyu .

Podemos ejecutar la prueba U de Mann-Whitney con nuestros datos de prueba de la Tabla 4-1,

---

```
de scipy.stats importar mannwhitneyu
u,p = mannwhitneyu(a,b)
print("U=%0.5f, p=%0.5f" % (u,p))
```

---

con a y b como usamos anteriormente para la prueba t. Esto nos da ( $U = 997.00000$ ,  $p = 0.04058$ ). El valor p está apenas por debajo del umbral mínimo de 0.05.

Las medias de a y b son 85 y 82, respectivamente. ¿Qué sucede con los valores p si hacemos que el valor medio de b sea 83 u 81? Cambiar la media de b significa cambiar el primer argumento a np.random.normal. Hacer esto nos da la Tabla 4-2, donde he incluido todos los resultados para que estén completos.

Tabla 4-2: Resultados de la prueba U de Mann-Whitney y de la prueba t para la prueba simulada Puntuaciones con diferentes medias ( $n_1=n_2=50$ )

Medio	Mann-Whitney U 85	prueba t
vs. 83 ( $U=1104.50000$ , $p=0.15839$ ) ( $t=1.66543$ , $p=0.09959$ )	85 vs. 82 ( $U=997.00000$ , $p=0.04058$ )	
( $t=2.40234$ , $p=0.01852$ )	85 vs. .81 ( $U=883.50000$ , $p=0.00575$ ) ( $t=3.13925$ , $p=0.00234$ )	

La tabla 4-2 debería tener sentido para nosotros. Cuando las medias son cercanas, es más difícil distinguirlas, por lo que esperamos valores p mayores. Recordemos como tenemos

sólo 50 muestras en cada grupo. A medida que aumenta la diferencia entre las medias, los valores p disminuyen. Una diferencia de tres en las medias conduce a valores p apenas significativos. Cuando la diferencia es aún mayor, los valores p se vuelven verdaderamente significativos (nuevamente, como esperábamos).

El análisis anterior plantea la pregunta: por una pequeña diferencia en las medias entre los dos grupos, ¿cómo cambian los valores p en función del tamaño de la muestra?

La Figura 4-8 muestra el valor p (media  $\pm$  error estándar) en 25 ejecuciones tanto para la prueba U de Mann-Whitney como para la prueba t en función del tamaño de la muestra para el caso en el que las medias son 85 y 84.

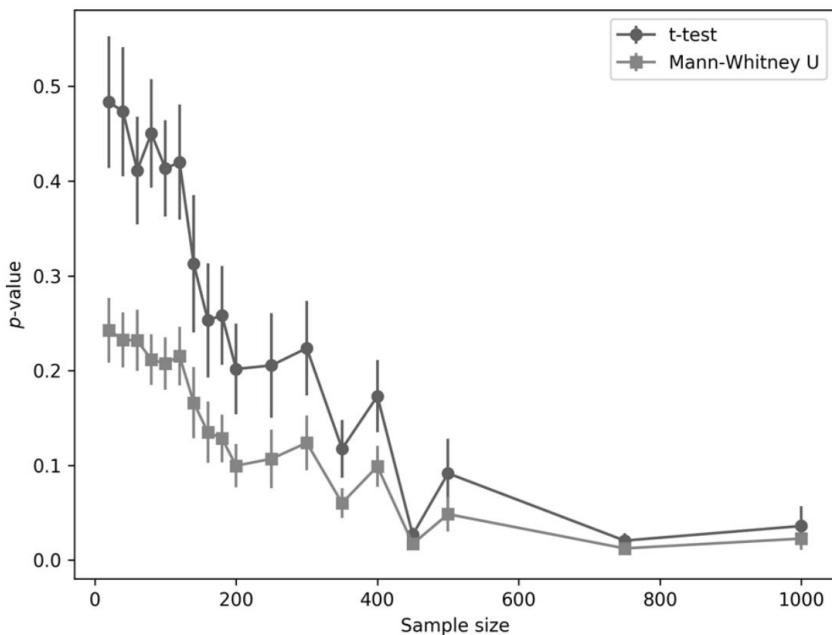


Figura 4-8: Valor p medio en función del tamaño de la muestra para una diferencia en las medias muestrales de uno,  $x_1 = 85$ ,  $x_2 = 84$

Los conjuntos de datos pequeños hacen difícil diferenciar entre casos cuando la diferencia en las medias es pequeña. También vemos que tamaños de muestra más grandes revelan la diferencia, independientemente de la prueba. Es interesante que en la figura 4-8, el valor p de Mann-Whitney Up sea menor que el de la prueba t, aunque los datos subyacentes se distribuyen normalmente. La sabiduría convencional afirma que normalmente es al revés.

La figura 4-8 es una lección práctica sobre el poder de las pruebas de muestras grandes para detectar diferencias reales. Cuando el tamaño de la muestra es lo suficientemente grande, una diferencia débil se vuelve significativa. Sin embargo, debemos equilibrar esto con el tamaño del efecto.

Cuando tenemos 1.000 muestras en cada grupo, tenemos un valor p estadísticamente significativo, pero también tenemos una d de Cohen de aproximadamente 0,13, lo que indica un efecto débil.

Un estudio de muestra grande podría encontrar un efecto significativo que sea tan débil que prácticamente carezca de significado.

## Resumen

Este capítulo abordó los aspectos clave de las estadísticas que encontrará durante su estancia en el mundo del aprendizaje profundo. Específicamente, aprendimos sobre diferentes tipos de datos y cómo garantizar que los datos sean útiles para construir modelos. Luego aprendimos sobre estadísticas resumidas y vimos ejemplos que las usaban para ayudarnos a comprender un conjunto de datos. Comprender nuestros datos es clave para un aprendizaje profundo exitoso. Investigamos los diferentes tipos de medias, aprendimos sobre medidas de variación y vimos la utilidad de visualizar los datos mediante diagramas de caja.

Los datos faltantes son una pesadilla para el aprendizaje profundo. En este capítulo, investigamos cómo compensar los datos faltantes. A continuación, analizamos la correlación, cómo detectar y medir las relaciones entre elementos de un conjunto de datos. Finalmente, introdujimos la prueba de hipótesis. Restringiéndonos al escenario más probable que encontraremos en el aprendizaje profundo, aprendimos cómo aplicar tanto la prueba t como la prueba U de Mann-Whitney. La prueba de hipótesis nos presentó el valor p. Vimos ejemplos y discutimos cómo interpretarlo correctamente.

En el próximo capítulo dejaremos atrás la estadística y nos sumergiremos de lleno en el mundo del álgebra lineal. El álgebra lineal es la forma en que implementamos redes neuronales.

# 5

## ÁLGEBRA LINEAL



Formalmente, el álgebra lineal es el estudio de ecuaciones lineales, en las que la potencia más alta de la variable es uno. Sin embargo, para nuestros propósitos, el álgebra lineal se refiere a conceptos multidimensionales. objetos matemáticos, como vectores y matrices, y operaciones sobre ellos. Así es como se aplica normalmente el álgebra lineal en el aprendizaje profundo y cómo se manipulan los datos en programas que implementan el aprendizaje profundo. algoritmos. Al hacer esta distinción, estamos desperdiando una enorme cantidad de matemáticas fascinantes, pero como nuestro objetivo es comprender las matemáticas utilizado y aplicado en el aprendizaje profundo, es de esperar que podamos sé perdonado.

En este capítulo, presentaré los objetos utilizados en el aprendizaje profundo, específicamente escalares, vectores, matrices y tensores. Como veremos, todos estos objetos son en realidad tensores de varios órdenes. Analizaremos los tensores desde una perspectiva matemática y notacional y luego experimentaremos con ellos usando NumPy. NumPy fue diseñado explícitamente para agregar matrices multidimensionales

a Python, y son buenos análogos, aunque incompletos, de los objetos matemáticos con los que trabajaremos en este capítulo.

Pasaremos la mayor parte del capítulo aprendiendo cómo hacer aritmética con tensores, lo cual es de fundamental importancia en el aprendizaje profundo. La mayor parte del esfuerzo para implementar kits de herramientas de aprendizaje profundo de alto rendimiento implica encontrar formas de hacer aritmética con tensores de la manera más eficiente posible.

## Escalares, vectores, matrices y tensores

Presentemos a nuestro elenco de personajes. Los relacionaré con variables de Python y matrices NumPy para mostrar cómo implementaremos estos objetos en el código. Luego presentaré un práctico mapeo conceptual entre tensores y geometría.

### escalares

Incluso si no estás familiarizado con la palabra, sabes qué es un escalar desde el día en que aprendiste a contar. Un escalar es simplemente un número, como 7, 42 o .

En las expresiones, usaremos  $x$  para indicar un escalar, es decir, la notación ordinaria utilizada para las variables. Para una computadora, un escalar es una variable numérica simple:

---

```
>>> s = 66
>>> s
66
```

---

### Vectores

Un vector es una matriz 1D de números. Matemáticamente, un vector tiene una orientación, ya sea horizontal o vertical. Si es horizontal, es un vector de fila. Por ejemplo,

$$x = [ x_0 x_1 x_2 ] \quad (5.1)$$

es un vector fila de tres elementos o componentes. Tenga en cuenta que usaremos  $x$ , una letra minúscula en negrita, para indicar un vector.

Matemáticamente, generalmente se supone que los vectores son vectores de columna,

$$y = \begin{matrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{matrix} \quad (5.2)$$

donde  $y$  tiene cuatro componentes, lo que lo convierte en un vector de cuatro dimensiones (4D). Observe que en la ecuación 5.1 usamos corchetes, mientras que en la ecuación 5.2 usamos paréntesis. Cualquiera de las notaciones es aceptable.

En código, normalmente implementamos vectores como matrices 1D:

---

```
>>> importar numpy como
np >>> x = np.array([1,2,3])
>>> imprimir(x)
[1 2 3]
>>> imprimir(x.reshape((3,1) ))
[[1]
 [2]
 [3]]
```

---

Aquí, hemos usado remodelar para convertir el vector fila de tres elementos en un vector columna de tres filas y una columna.

Las componentes de un vector a menudo se interpretan como longitudes a lo largo de un conjunto de ejes de coordenadas. Por ejemplo, se podría utilizar un vector de tres componentes para representar un punto en el espacio 3D. En este vector,

$$\mathbf{x} = [x,y,z]$$

$x$  podría ser la longitud a lo largo del eje  $x$ , y la longitud a lo largo del eje  $y$  y  $z$  la longitud a lo largo del eje  $z$ . Estas son las coordenadas cartesianas y sirven para identificar de forma única todos los puntos en el espacio 3D.

Sin embargo, en el aprendizaje profundo y en el aprendizaje automático en general, los componentes de un vector a menudo no están relacionados entre sí en un sentido geométrico estricto. Más bien, se utilizan para representar características, cualidades de alguna muestra que el modelo utilizará para intentar llegar a un resultado útil, como una etiqueta de clase o un valor de regresión. Dicho esto, el vector que representa el conjunto de características, llamado vector de características, a veces se considera geométricamente. Por ejemplo, algunos modelos de aprendizaje automático, como los  $k$  vecinos más cercanos, interpretan que el vector representa alguna coordenada en el espacio geométrico.

A menudo escucharás a personas de aprendizaje profundo discutir el espacio de características de un problema. El espacio de características se refiere al conjunto de posibles entradas. El conjunto de entrenamiento de un modelo debe representar con precisión el espacio de características de las posibles entradas que encontrará el modelo cuando se utilice. En este sentido, el vector de características es un punto, una ubicación en este espacio de  $n$  dimensiones donde  $n$  es el número de características en el vector de características.

## matrices

Una matriz es una matriz 2D de números:

$$\text{Una} = \begin{matrix} & a_{00} & a_{01} & a_{02} & a_{03} \\ a_{00} & a_{10} & a_{11} & a_{12} & a_{13} \\ a_{01} & a_{20} & a_{21} & a_{22} & a_{23} \end{matrix}$$

Los elementos de A están subíndices por el número de fila y el número de columna. La matriz A tiene tres filas y cuatro columnas, por lo que decimos que es una matriz de  $3 \times 4$ , donde  $3 \times 4$  es el orden de la matriz. Observe que A usa subíndices que comienzan con 0. Los textos matemáticos a menudo comienzan con 1, pero cada vez más usan 0 para que no haya un desplazamiento entre la notación matemática y la representación informática de la matriz. Tenga en cuenta también que usaremos A, una letra mayúscula en negrita, para referirnos a una matriz.

En el código, las matrices se representan como matrices 2D:

---

```
>>> A = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>>
```

`imprimir(A) [[1 2 3] [4 5 6] [7 8 9]] >>>`

---

```
print(np.arange(12).reshape((3,4))) [[ 0  1  2  3] [ 4  5  6  7] [ 8  9  10  11]]
```

Para obtener el elemento a12 de A en Python, escribimos A[1,2]. Observe que cuando imprimimos las matrices, había un [ y ] adicional alrededor de ellas. NumPy usa estos corchetes para indicar que la matriz 2D puede considerarse como un vector de fila en el que cada elemento es en sí mismo un vector. En Python, esto significa que una matriz puede considerarse como una lista de sublistas en la que cada sublista tiene la misma longitud. Por supuesto, así es exactamente como definimos A para empezar.

Podemos pensar en los vectores como matrices con una sola fila o columna. Un vector columna con tres elementos es una matriz de  $3 \times 1$ : tiene tres filas y una columna. De manera similar, un vector fila de cuatro elementos actúa como una matriz de  $1 \times 4$ : tiene una fila y cuatro columnas. Haremos uso de esta observación más adelante.

## Tensores

Un escalar no tiene dimensiones, un vector tiene una y una matriz tiene dos. Como puede sospechar, no es necesario que nos detengamos allí. Un objeto matemático con más de dos dimensiones se denomina coloquialmente tensor. Cuando sea necesario, representaremos tensores como este: T, como una letra mayúscula sans serif.

El número de dimensiones que tiene un tensor define su orden, que no debe confundirse con el orden de una matriz. Un tensor 3D tiene orden 3. Una matriz es un tensor de orden 2. Un vector es un tensor de orden 1 y un escalar es un tensor de orden 0. Cuando analicemos el flujo de datos a través de una red neuronal profunda en el Capítulo 9, veremos que muchos conjuntos de herramientas utilizan tensores de orden 4 (o más).

En Python, las matrices NumPy con tres o más dimensiones se utilizan para implementar tensores. Por ejemplo, podemos definir un tensor de orden 3 en Python como se muestra a continuación:

---

```
>>> t = np.arange(36).reshape((3,3,4)) >>> print(t)
[[[ 0  1  2  3] [ 4 5
 6 7] [ 8 9 10 11]]]
```

```
[[12 13 14 15] [16
 17 18 19] [20 21
 22 23]]]
```

```
[[24 25 26 27] [28
 29 30 31] [32 33
 34 35]]]
```

---

Aquí usamos `np.arange` para definir `t` como un vector de 36 elementos que contienen los números 0 . . . 35. Luego, inmediatamente transformamos el vector en un tensor de  $3 \times 3 \times 4$  elementos ( $3 \times 3 \times 4 = 36$ ). Una forma de pensar en un decena de  $3 \times 3 \times 4$  es que contiene una pila de tres imágenes de  $3 \times 4$ . Si tenemos esto en cuenta, las siguientes afirmaciones tienen sentido:

```
>>> imprimir(t[0])
[[ 0  1  2  3] [ 4 5 6
 7] [ 8 9 10 11]]
>>> imprimir(t[0,1])
[4 5 6 7]>> >
```

```
imprimir(t[0,1,2]) 6
```

---

Al solicitar `t[0]` se devolverá la primera imagen de  $3 \times 4$  de la pila. Al solicitar `t[0,1]`, entonces, se debería devolver la segunda fila de la primera imagen, lo cual ocurre. Finalmente, llegamos a un elemento individual de `t` preguntando por el número de imagen (0), el número de fila (1) y el elemento de esa fila (2).

Asignar las dimensiones de un tensor a colecciones sucesivamente más pequeñas de algo es una forma práctica de tener en cuenta el significado de las dimensiones. Por ejemplo, podemos definir un tensor de orden 5 así:

---

```
>>> w = np.zeros((9,9,9,9,9)) >>>
w[4,1,2,0,1]
0.0
```

---

Pero, ¿qué significa pedir  $w[4,1,2,0,1]$ ? El significado exacto depende de la aplicación. Por ejemplo, podríamos pensar que  $w$  representa una estantería. El primer índice selecciona el estante y el segundo selecciona el libro en el estante. Luego, el tercer índice selecciona la página dentro del libro y el cuarto selecciona la línea de la página. El índice final selecciona la palabra en la línea. Por lo tanto,  $w[4,1,2,0,1]$  está pidiendo la segunda palabra de la primera línea de la tercera página del segundo libro en el quinto estante de la estantería, entendido leyendo los índices de derecha a izquierda.

La analogía de la estantería tiene sus limitaciones. Los arreglos NumPy tienen dimensiones fijas, lo que significa que si  $w$  es una estantería, hay nueve estantes y cada estante tiene exactamente nueve libros. Asimismo, cada libro tiene exactamente nueve páginas y cada página tiene nueve líneas. Finalmente, cada línea tiene exactamente nueve palabras. Las matrices NumPy normalmente usan memoria contigua en la computadora, por lo que el tamaño de cada dimensión se fija cuando se define la matriz. Hacerlo y seleccionar el tipo de datos específico, como un entero sin signo, hace que ubicar un elemento de la matriz sea una operación de indexación usando una fórmula simple para calcular un desplazamiento desde una dirección de memoria base. Esto es lo que hace que las matrices NumPy sean mucho más rápidas que las listas de Python.

Cualquier tensor de orden menor que  $n$  se puede representar como un tensor de orden  $n$  proporcionando las dimensiones faltantes de longitud uno. Vimos un ejemplo de esto arriba cuando dije que un vector de componentes  $m$  podría considerarse como una matriz de  $1 \times m$  o  $m \times 1$ . El tensor de orden 1 (el vector) se convierte en un tensor de orden 2 (matriz) agregando una dimensión faltante de longitud uno.

Como ejemplo extremo, podemos tratar un escalar (tensor de orden 0) como un tensor de orden 5, así:

---

```
>>> t = np.array(42).reshape((1,1,1,1,1)) >>>
imprimir(t)
[[[[[42]]]] >>>
t.shape (1, 1,
1, 1, 1) >>>
t[0,0,0,0,0]
42
```

---

Aquí, transformamos el escalar 42 en un tensor de orden 5 (una matriz [5D] de cinco dimensiones) con una longitud uno en cada eje. Observe que NumPy nos dice que la  $t$  de diez tipos tiene cinco dimensiones con  $[[[[[y]]]]]$  alrededor de 42. Preguntar por la forma de  $t$  confirma que es un tensor 5D. Finalmente, como tensor, podemos obtener el valor del único elemento que contiene especificando todas las dimensiones con  $t[0,0,0,0,0]$ . A menudo usaremos este truco de agregar nuevas dimensiones de longitud uno. De hecho, en NumPy, hay una manera de hacer esto directamente, que verá cuando utilice kits de herramientas de aprendizaje profundo:

---

```
>>> t = np.array([[1,2,3],[4,5,6]]) >>>
imprimir(t) [[1
2 3] [4 5
6]] >>> w
= t[np.nuevaje,:,:]
```

---

```
>>> w.forma
(1, 2, 3)
>>> imprimir(w)
[[[1 2 3]
 [4 5 6]]]
```

---

Aquí, hemos convertido  $t$ , un tensor de orden 2 (una matriz), en un tensor de orden 3 usando `np.newaxis` para crear un nuevo eje de longitud uno. Por eso `w.shape` devuelve `(1,2,3)` y no `(2,3)`, como lo haría para  $t$ .

Hay analogías entre tensores de hasta orden 3 y geometría que son útiles para visualizar las relaciones entre los diferentes órdenes:

Orden (dimensiones)	Nombre del tensor	Nombre geométrico
0	Escalar	Punto
1	Vector	Línea
2	Matriz	Avión
3	Tensor	Volumen

Aviso, usé tensor en su sentido común en la tabla. Parece ser que no hay un nombre estandarizado para un tensor de orden 3.

En esta sección, definimos los objetos matemáticos del aprendizaje profundo en relación con matrices multidimensionales, ya que así es como se implementan en código. Hemos desperdiciado muchas matemáticas al hacer esto, pero hemos preservado lo que necesitamos para comprender el aprendizaje profundo. sigamos adelante ahora y vea cómo usar tensores en expresiones.

## Aritmética con tensores

El propósito de esta sección es detallar las operaciones con tensores, con especial énfasis en tensores de orden 1 (vectores) y orden 2 (matrices). Asumiremos que las operaciones con escalares están bien controladas en este punto.

Comenzaremos con lo que llamo operaciones de matriz, con lo que me refiero a Operaciones de elementos que los kits de herramientas como NumPy realizan en matrices de todas las dimensiones. Luego pasaremos a operaciones particulares de vectores. Esto establece el escenario para el tema crítico de la multiplicación de matrices. Finalmente, discutiremos matrices de bloques.

### Operaciones de matriz

La forma en que hemos utilizado el kit de herramientas NumPy hasta ahora nos ha demostrado que todas las operaciones aritméticas escalares normales se traducen directamente en el mundo de los arreglos multidimensionales. Esto incluye operaciones estándar como suma, resta, multiplicación, división y exponentiación, así como la aplicación de funciones a una matriz. En todos estos casos, la operación escalar se aplica elemento por elemento a cada elemento de la matriz.

Los ejemplos aquí establecerán el tono para el resto de esta sección y también nos permitirán explorar algunos NumPy reglas de transmisión que aún no hemos anunciado.

---

Primero definamos algunas matrices con las que trabajar:

---

```
>>> a = np.array([[1,2,3],[4,5,6]]) >>> b =
np.array([[7,8,9],[10,11, 12]]) >>> c =
np.array([10,100,1000]) >>> d =
np.array([10,11]) >>> imprimir(a)
[[1 2 3] [4 5 6]]
>>>
imprimir(b)
[[ 7 8 9] [10 11
12]] >>>
imprimir(c)
[ 10 100 1000]
>>> imprimir(d) [10
11]
```

---

La aritmética por elementos es sencilla para matrices con dimensiones ese partido:

---

```
>>> imprimir(a+b)
[[ 8 10 12] [14
16 18]] >>>
imprimir(ab) [[-6 -6
-6] [-6 -6 -6]]
>>> imprimir
(a*b) [[ 7 16 27]
[40 55 72]]
>>> imprimir(a/
b) [[0.14285714
0.25 [0.4 >>>
0,33333333] ]]
0,45454545 0,5
imprimir(b**a) [[ 64
729] [ 10000 161051 2985984]]
```

---

Todos estos resultados son bastante fáciles de interpretar; NumPy aplica la operación deseada a los elementos correspondientes de cada matriz. La multiplicación de elementos en dos matrices (  $a \circ b$  ) a menudo se conoce como producto de Hadamard. (Encontrará este término de vez en cuando en la literatura sobre aprendizaje profundo).

El kit de herramientas NumPy extiende la idea de operaciones por elementos a lo que llama transmisión. Al transmitir, NumPy aplica reglas, que veremos a través de ejemplos, donde una matriz se pasa sobre otra para producir una salida significativa.

Ya nos hemos encontrado con una forma de transmisión cuando operamos en una matriz con un escalar. En ese caso, el valor escalar se transmitió a cada valor de la matriz.

Para nuestro primer ejemplo, aunque  $a$  es una matriz de  $2 \times 3$ , NumPy permite operaciones con  $c$ , un vector de tres componentes, aplicando radiodifusión:

---

```
>>> imprimir(a+c)
[[ 11 102 1003] [ 14
 105 1006]] >>>
imprimir(c*a) [[ 10
200 3000] [ 40 500
6000]] >>> imprimir(a/
c ) [[0,1 0,02
0,003] [0,4 0,05 0,006]]
```

---

Aquí, el vector de tres componentes,  $c$ , se difundió sobre las filas de la matriz de  $2 \times 3$ ,  $a$ . NumPy reconoció que las últimas dimensiones de  $a$  y  $c$  eran tres, por lo que el vector podría pasarse sobre la matriz para producir el resultado dado. Al observar el código de aprendizaje profundo, gran parte del cual está en Python, verá situaciones como esta. A veces, es necesario pensar un poco, junto con algo de experimentación en el indicador de Python, para comprender lo que está sucediendo.

¿Podemos transmitir  $d$ , un vector de dos componentes, sobre  $a$ , una matriz de  $2 \times 3$ ? Si intentamos hacerlo de la misma manera que transmitimos  $c$  sobre  $a$ , fallaremos:

---

```
>>> imprimir(a+d)
Rastreo (llamadas recientes más última):
  Archivo "<stdin>", línea 1, en <módulo>
ValueError: los operandos no se pudieron transmitir junto con las formas (2,3) (2,)
```

---

Sin embargo, las reglas de transmisión de NumPy se adaptan a dimensiones de longitud uno. La forma de  $d$  es 2; es un vector de dos elementos. Si remodelamos  $d$  para que sea una matriz 2D con forma  $2 \times 1$ , le daremos a NumPy lo que necesita:

---

```
>>> d = d.reformar((2,1)) >>>
d.forma (2, 1)
>>>
imprimir(a+d) [[11
12 13] [15 16
17]]
```

---

Ahora vemos que Numpy ha agregado  $d$  en las columnas de  $a$ .

Volvamos al mundo de las matemáticas y veamos las operaciones en vectores.

### Operaciones con

vectores Los vectores se representan en código como una colección de números que pueden interpretarse como valores a lo largo de un conjunto de ejes de coordenadas. Aquí definiremos varias operaciones que son exclusivas de los vectores.

### Magnitud

Geométricamente, podemos entender que los vectores tienen una dirección y una longitud. A menudo se dibujan como flechas y veremos un ejemplo de una gráfica vectorial en el Capítulo 6. La gente habla de la longitud de un vector como su magnitud. Por lo tanto, la primera operación vectorial que consideraremos es calcular su magnitud.

Para un vector,  $x$ , con  $n$  componentes, la fórmula para su magnitud es

$$x = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} \quad (5.3)$$

En la ecuación 5.3, las barras verticales dobles alrededor del vector representan su magnitud. Aquí también verás a menudo que la gente usa barras individuales. Las barras individuales también se utilizan para valores absolutos; Por lo general, confiamos en el contexto para diferenciar entre los dos.

¿De dónde surgió la ecuación 5.3? Considere un vector en 2D,  $x = (x, y)$ . Si  $x$  e  $y$  son longitudes a lo largo de los ejes  $x$  e  $y$ , respectivamente, vemos que  $x$  e  $y$  forman los lados de un triángulo rectángulo. La longitud de la hipotenusa de este triángulo rectángulo es la longitud del vector. Por lo tanto, según Pitágoras, y los babilonios mucho antes que él, esta longitud es  $\sqrt{x^2 + y^2}$ , que, generalizada a  $n$  dimensiones, se convierte en la ecuación 5.3.

### Vectores unitarios

Ahora que podemos calcular la magnitud de un vector, podemos introducir una forma útil de vector conocida como vector unitario. Si dividimos los componentes de un vector por su magnitud, nos queda un vector que apunta en la misma dirección que el vector original pero tiene una magnitud de uno. Este es el vector unitario. Para un vector,  $v$ , el vector unitario en la misma dirección es

$$v = \frac{v}{|v|}$$

donde el sombrero sobre el vector sirve para identificarlo como vector unitario. Veamos un ejemplo concreto. Nuestro vector de ejemplo es  $v = (2, -4, 3)$ . Por lo tanto, el vector unitario en la misma dirección que  $v$  es

$$v = \frac{(2, -4, 3)}{\sqrt{2^2 + (-4)^2 + 3^2}} = (2, -4, 3) / \sqrt{29} \approx (0.3714, -0.7428, 0.5571)$$

En código, calculamos el vector unitario de la siguiente manera:

---

```
>>> v = np.array((2, -4, 3))
>>> u = v / np.sqrt((v*v).sum())
>>>
imprimir(u) [ 0.37139068 -0.74278135 0.55708601 ]
```

---

Aquí, aprovechamos el hecho de que para elevar al cuadrado cada elemento de  $v$ , lo multiplicamos por sí mismo, por elementos, y luego sumamos los componentes llamando a  $\text{sum}$  para obtener la magnitud al cuadrado.

Transposición de

vectores Mencionamos anteriormente que los vectores de fila pueden considerarse como matrices de  $1 \times n$ , mientras que los vectores de columna son matrices de  $n \times 1$ . El acto de cambiar un vector fila en un vector columna y viceversa se conoce como transposición. Veremos en el Capítulo 6 que la transpuesta también se aplica a las matrices. Notacionalmente, denotamos la transpuesta vectorial de  $y$  como  $y^T$ . Por lo tanto, tenemos

$$x = [x_0 \ x_1 \ x_2]$$

$$\begin{matrix} & x_0 \\ x^T = & x_1 \\ & x_2 \end{matrix}$$

$$\begin{matrix} & y_0 \\ y = & y_1 \\ & y_2 \end{matrix}$$

$$y^T = [y_0 \ y_1 \ y_2]$$

$$(z^T)^T = z$$

Por supuesto, no estamos limitados a sólo tres componentes.

En código, transponemos vectores de varias maneras. Como vimos arriba, podemos usar remodelar para remodelar el vector en una matriz de  $1 \times n$  o  $n \times 1$ . También podemos llamar al método de transposición en el vector, con cierto cuidado, o usar la taquigrafía de transposición. Veamos ejemplos de todos estos enfoques. Primero, definamos un vector NumPy y veamos cómo la remodelación lo convierte en un vector de columna de  $3 \times 1$  y un vector de fila de  $1 \times 3$ , a diferencia de un vector simple de tres elementos:

---

```
>>> v = np.array([1,2,3])
>>>

imprimir(v) [1 2 3] >>>

imprimir(v.reshape((3,1))) [[1]
[ 2] [3]] >>> imprimir(v.reformar((1,3))) [[1 2 3]]
```

---

Observe la diferencia entre la primera impresión (`v`) y la última después de llamar a `reshape((1,3))`. La salida ahora tiene un conjunto adicional de corchetes alrededor para indicar la dimensión principal de uno.

A continuación, aplicamos la operación de transposición en `v`:

---

```
>>> imprimir(v.transponer())
[1 2 3]
>>> imprimir(vT)
[1 2 3]
```

---

Aquí vemos que llamar a transpuesta o `T` no cambia nada acerca de `v`. Esto se debe a que la forma de `v` es simplemente 3, no  $(1,3)$  o  $(3,1)$ . Si modificamos explícitamente `v` para que sea una matriz de  $1 \times 3$ , vemos que la transposición y `T` tienen el efecto deseado:

---

```
>>> v = v.reformar((1,3))
>>> imprimir(v.transponer())
[[1]
 [2]
 [3]]
>>> imprimir(vT)
[[1]

 [2]
 [3]]
```

---

Aquí, `v` pasa de ser un vector fila a un vector columna, como esperábamos. La lección, entonces, es tener cuidado con la dimensionalidad real de los vectores en el código NumPy. La mayoría de las veces podemos ser descuidados, pero a veces debemos ser explícitos y preocuparnos por la distinción entre vectores simples, vectores de fila y vectores de columna.

### Producto

interno Quizás la operación vectorial más común sea el producto interno o, como se le llama frecuentemente, el producto escalar. Notacionalmente, el producto interno entre dos vectores se escribe como

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}, \mathbf{b} = \mathbf{a} \mathbf{b}$$

$$= \sum_{k=0}^{n-1} a_k b_k \quad (5.4)$$

$$= \mathbf{a} \mathbf{b} \text{ porque} \quad (5.5)$$

Aquí está el ángulo entre los dos vectores si se interpretan geométricamente. El resultado del producto interno es un escalar. La notación `a, b` se ve con frecuencia, aunque la notación de punto `a • b` parece más común en la literatura sobre aprendizaje profundo. La notación de multiplicación de matrices `a b` indica explícitamente cómo calcular el producto interno, pero esperaremos hasta discutir la matriz.

multiplicación para explicar su significado. Por el momento, la suma nos dice lo que necesitamos saber: el producto interno de dos vectores de longitud  $n$  es la suma de los productos de los  $n$  componentes.

El producto interno de un vector consigo mismo es la magnitud al cuadrado:

$$\mathbf{a} \cdot \mathbf{a} = |\mathbf{a}|^2$$

El producto interno es conmutativo,

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$$

y distributivo,

$$\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}$$

pero no asociativo, ya que la salida del primer producto interno es un escalar, no un vector, y multiplicar un vector por un escalar no es un producto interno.

Finalmente, observe que el producto interno es cero cuando el ángulo entre los vectores es de 90 grados; esto se debe a que  $\cos$  es cero (Ecuación 5.5). Esto significa que los dos vectores son perpendiculares u ortogonales entre sí.

Veamos algunos ejemplos del producto interno. Primero, seremos literales e implementaremos la Ecuación 5.4 explícitamente:

---

```
>>> a = np.array([1,2,3,4]) >>>
b = np.array([5,6,7,8]) >>> def
interior(a,b): s = 0,0
...
... para i en el rango(len(a)): s
...     += a[i]*b[i]
... devoluciones
...
>>> interior(a,b)
70.0
```

---

Sin embargo, dado que  $a$  y  $b$  son matrices NumPy, sabemos que podemos ser más eficientes:

---

```
>>> (a*b).sum()
70
```

---

O, probablemente lo más eficiente de todo, dejaremos que NumPy lo haga por nosotros usando `np.dot`:

---

```
>>> np.punto(a,b)
70
```

---

Verás `np.dot` con frecuencia en el código de aprendizaje profundo. Puede hacer más que calcular el producto interno, como veremos a continuación.

La ecuación 5.5 nos dice que el ángulo entre dos vectores es

$$\theta = \cos^{-1} \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|}$$

En el código, esto podría calcularse como

---

```
>>> A = np.sqrt(np.punto(a,a))
>>> B = np.sqrt(np.punto(b,b)) >>> t
= np.arccos(np.punto(a,b)/(A*B)) >>> t*(180/
np.pi)
14.335170291600924
```

---

Esto nos dice que el ángulo entre  $a$  y  $b$  es aproximadamente  $14^\circ$  después de convertir  $t$  en radianes.

Si consideramos vectores en el espacio 3D, vemos que el producto escalar entre vectores ortogonales es cero, lo que implica que el ángulo entre ellos es de  $90^\circ$ :

---

```
>>> a = np.matriz([1,0,0]) >>> b
= np.matriz([0,1,0]) >>>
np.punto(a,b)
0

>>> t = np.arccos(0) >>>
t*(180/np.pi)
90.0
```

---

Esto es cierto porque  $a$  es un vector unitario a lo largo del eje  $x$ ,  $b$  es un vector unitario a lo largo del eje  $y$  y sabemos que hay un ángulo recto entre ellos.

Con el producto interno en nuestro conjunto de herramientas, veamos cómo podemos usarlo para proyectar un vector sobre otro.

### Proyección

La proyección de un vector sobre otro calcula la cantidad del primer vector que está en la dirección del segundo. La proyección de  $a$  sobre  $b$  es

$$\text{proyecto } a = \frac{a \cdot b}{b^2} b$$

La Figura 5-1 muestra gráficamente lo que significa proyección para vectores 2D.

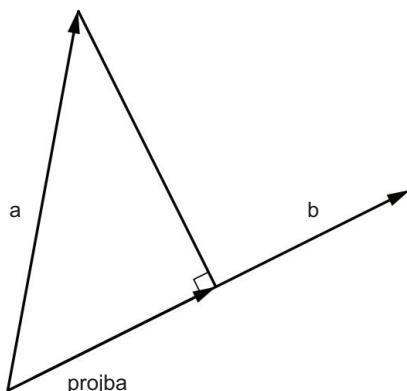


Figura 5-1: Una representación gráfica de la proyección de  $a$  sobre  $b$  en 2D

La proyección encuentra la componente de  $a$  en la dirección de  $b$ . Tenga en cuenta el producto interno de  $a$  sobre  $b$  no es lo mismo que la proyección de  $b$  sobre  $a$ .

Como usamos un producto interno en el numerador, podemos ver que la proyección de un vector sobre otro vector que es ortogonal a él es cero. Ninguna componente del primer vector está en la dirección del segundo. Piensa de nuevo en los ejes  $x$  y  $y$ . La única razón por la que usamos coordenadas cartesianas es porque los dos ejes, o tres en el espacio 3D, son todos ortogonales entre sí; ninguna parte de uno está en dirección a los demás. Esto nos permite especificar cualquier punto y el vector desde el origen hasta ese punto, especificando los componentes a lo largo de estos ejes. Veremos esta división de un objeto en componentes mutuamente ortogonales más adelante, cuando analicemos los vectores propios y el PCA en el Capítulo 6.

En código, calcular la proyección es sencillo:

---

```
>>> a = np.array([1,1])
>>> b = np.array([1,0])
>>> p = (np.punto(a,b)/np.punto(b ,b))*b
>>>

imprimir(p) [1. 0.] >>> c =
np.array([-1,1]) >>> p = (np.dot(c,b)/
np.dot(b,b))*b
>>> imprimir( p) [-1. -0.]
```

---

En el primer ejemplo,  $a$  apunta en la dirección  $45^\circ$  hacia arriba desde el eje  $x$ , mientras que  $b$  apunta a lo largo del eje  $x$ . Entonces esperaríamos que la proyección de  $a$  fuera a lo largo del eje  $x$ , que es  $(p)$ . En el segundo ejemplo,  $c$  apunta en la dirección  $135^\circ = 90^\circ + 45^\circ$  desde el eje  $x$ . Por lo tanto, esperaríamos que la componente de  $c$  a lo largo de  $b$  estuviera a lo largo del eje  $x$  pero en la dirección opuesta a  $b$ , que es lo que es.

### NOTA

Proyectar  $c$  a lo largo de  $b$  arrojó un componente del eje  $y$  de  $-0$ . El signo negativo es una peculiaridad de la representación IEEE 754 utilizada para números de punto flotante. El significado (mantisa) de la representación interna es cero, pero el signo aún se puede especificar, lo que lleva a una salida de cero negativo de vez en cuando. Para obtener una explicación detallada de los formatos de números de computadora, incluido el punto flotante, consulte mi libro *Numbers and Computers* (Springer-Verlag, 2017).

Pasemos ahora a considerar el producto exterior de dos vectores.

### Producto exterior

El producto interno de dos vectores devolvió un valor escalar. En cambio, el producto exterior de dos vectores devuelve una matriz. Tenga en cuenta que, a diferencia del producto interior, el producto exterior no requiere que los dos vectores tengan el mismo número de componentes. Específicamente, para los vectores  $a$  de  $m$  componentes y  $b$  de  $n$  componentes, el producto externo es la matriz formada multiplicando cada elemento de  $a$  por cada elemento de  $b$ , como se muestra a continuación.

$$\begin{array}{c}
 & a_0 b_0 & a_0 b_1 & a_0 b_2 & a_0 b_{n-1} a_1 \\
 a \quad b = ab & = & a_1 b_0 & a_1 b_1 & b_2 \quad a_1 b_{n-1} \\
 & am-1b_0 & am-1b_1 & am-1b_2 & am-1b_{n-1}
 \end{array}$$

La notación  $ab$  es cómo calcular el producto exterior mediante la multiplicación de matrices. Observe que esta notación no es la misma que el producto interno,  $a \cdot b$ , y que supone que  $a$  y  $b$  son vectores columna. No se utiliza consistentemente ningún símbolo de operador para el producto exterior, principalmente porque se especifica fácilmente mediante la multiplicación de matrices y porque es menos común que el producto escalable. Sin embargo, parece ser el más utilizado cuando el producto exterior se presenta con un operador binario.

En el código, NumPy nos ha proporcionado amablemente una función de producto externo:

---

```
>>> a = np.array([1,2,3,4]) >>> b
= np.array([5,6,7,8]) >>>
np.punto(a,b)
70
>>> np.outer(a,b)
matriz([[ 5,  6,  7,  8], [10, 12,
14, 16], [15, 18, 21,
24], [20, 24, 28 , 32]])
```

---

Usamos  $a$  y  $b$  arriba cuando hablamos del producto interno. Como era de esperar, `np.dot` nos da una salida escalar para  $a \cdot b$ . Sin embargo, la función `np.outer` devuelve una matriz de  $4 \times 4$ , donde vemos que cada fila es el vector  $b$  multiplicado sucesivamente por cada elemento del vector  $a$ , primero 1, luego 2, luego 3 y finalmente 4. Por lo tanto, cada elemento de  $a$  ha multiplicado cada elemento de  $b$ . La matriz resultante es  $4 \times 4$  porque tanto  $a$  como  $b$  tienen cuatro elementos.

### EL PRODUCTO CARTESIANO

Existe un análogo directo entre el producto exterior de dos vectores y el producto cartesiano de dos conjuntos,  $A$  y  $B$ . El producto cartesiano es un nuevo conjunto, cada elemento del cual es uno de los posibles pares de elementos de  $A$  y  $B$ . Entonces, si  $A=\{1,2,3,4\}$  y  $B=\{5,6,7,8\}$ , el producto cartesiano se puede escribir como

$$\begin{aligned}
 \times = \{ & ( , ) \mid y \} (1, 5) (1, \\
 & 6) (1, 7) (1, 8) (2, 5) (2, \\
 & 6) (2, 7) (2, 8) (3, 5) \\
 = & (3, 6) (3, 7) (3, 8) \\
 & (4, 5) (4, 6) (4, 7) (4, 8)
 \end{aligned}$$

Aquí vemos que si reemplazamos cada entrada con el producto del par, obtenemos el producto vectorial correspondiente que vimos arriba con NumPy np.outer. Además, tenga en cuenta que  $\times$  se utiliza normalmente para el producto cartesiano cuando se trabaja con conjuntos.

La capacidad del producto externo para mezclar todas las combinaciones de sus entradas se ha utilizado en aprendizaje profundo para aplicaciones de filtrado colaborativo neuronal y respuesta visual a preguntas. Estas funciones las realizan redes avanzadas que hacen recomendaciones o responden preguntas de texto sobre una imagen. El producto exterior aparece como una mezcla de dos vectores de incrustación diferentes. Las incrustaciones son los vectores generados por las capas inferiores de una red, por ejemplo, la penúltima capa completamente conectada antes de la salida de la capa softmax de una red neuronal convolucional tradicional (CNN). Generalmente se considera que la capa de incrustación ha aprendido una nueva representación de la entrada de la red. Se puede considerar como un mapeo de entradas complejas, como imágenes, en un espacio reducido de varios cientos a varios miles de dimensiones.

### Producto

Cruzado Nuestro operador vector-vector final es el producto cruzado. Este operador solo está definido para el  $\mathbb{R}^3$ . El producto cruzado de  $a$  y  $b$  es un nuevo vector que es perpendicular al plano que contiene  $a$  y  $b$ . Tenga en cuenta que esto no implica que  $a$  y  $b$  sean perpendiculares. El producto vectorial se define como

$$\begin{aligned} a \times b &= |a||b| \sin(\theta) \\ &= (a_1 b_2 - a_2 b_1, a_0 b_2 - a_2 b_0, a_0 b_1 - a_1 b_0) \end{aligned} \tag{5.6}$$

donde  $n$  es un vector unitario y  $\theta$  es el ángulo entre  $a$  y  $b$ . La dirección de  $n$  está dada por la regla de la mano derecha. Con la mano derecha, apunte con el dedo índice en dirección a y con el dedo medio en dirección b. Luego, tu pulgar apuntará en la dirección de  $n$ . La ecuación 5.6 proporciona los componentes reales del vector del producto vectorial.

NumPy implementa el producto cruzado a través de np.cross:

---

```
>>> a = np.array([1,0,0]) >>>
b = np.array([0,1,0]) >>>
imprimir(np.cruz(a,b)) [0
1] >>>
c = np.array([1,1,0]) >>>
imprimir(np.cruz(a,c)) [0 0 1]
```

---

En el primer ejemplo, a apunta a lo largo del eje x y b a lo largo del eje y. Por lo tanto, esperamos que el producto vectorial sea perpendicular a estos ejes, y lo es: el producto vectorial apunta a lo largo del eje z. El segundo ejemplo muestra que no importa si a y b son perpendiculares entre sí. Aquí,

c está en un ángulo de  $45^\circ$  con respecto al eje x, pero a y c todavía están en el plano xy. Por lo tanto, el producto vectorial todavía está a lo largo del eje z.

La definición del producto cruzado implica el pecado. , mientras que el producto interno usa cos. El producto interno es cero cuando los dos vectores son ortogonales entre sí. El producto vectorial, por otro lado, es cero cuando los dos vectores están en la misma dirección y se maximiza cuando los vectores son perpendiculares. El segundo ejemplo de NumPy anterior funciona porque la magnitud de c es  $\sqrt{2}$  y  $\sin 45^\circ = \sqrt{2}/2 = 1/\sqrt{2}$ . Como resultado, los factores  $\sqrt{2}$  se cancelan para dejar una magnitud de 1 para el producto cruzado. porque a es una unidad vector.

El producto cruzado se usa ampliamente en física y otras ciencias, pero es menos A menudo se utiliza en aprendizaje profundo debido a su restricción al espacio 3D. No obstante, deberías estar familiarizado con él si vas a abordar la literatura sobre aprendizaje profundo.

Con esto concluye nuestra mirada a las operaciones vector-vector. Dejemos el mundo 1D y pasemos a considerar la operación más importante para todo el aprendizaje profundo: la multiplicación de matrices.

### Multiplicación de matrices

En la sección anterior, vimos cómo multiplicar dos vectores de varias maneras: producto de Hadamard, producto interno (punto), producto externo y producto vectorial.

En esta sección, investigaremos la multiplicación de matrices, recordando que los vectores de fila y columna son en sí mismos matrices con una fila o columna.

### Propiedades de la multiplicación

de matrices Definiremos la operación del producto matricial en breve, pero, antes de hacerlo, veamos las propiedades de la multiplicación de matrices. Sean A, B y C matrices. Luego, siguiendo la convención de álgebra de multiplicar símbolos colocándolos uno al lado del otro,

$$(AB)C = A(BC)$$

lo que significa que la multiplicación de matrices es asociativa. En segundo lugar, la multiplicación de matrices es distributiva:

$$A(B + C) = AB + CA \quad (5.7)$$

$$(A + B)C = AC + BC \quad (5.8)$$

Sin embargo, en general, la multiplicación de matrices no es comutativa:

$$AB \neq BA$$

Como puede ver en la Ecuación 5.8, la multiplicación de matrices sobre la suma desde la derecha produce un resultado diferente que la multiplicación de matrices sobre la suma desde la izquierda, como se muestra en la Ecuación 5.7. Esto explica por qué mostramos tanto la Ecuación 5.7 como la Ecuación 5.8; La multiplicación de matrices se puede realizar desde la izquierda o desde la derecha y el resultado será diferente.

### Cómo multiplicar dos

matrices Para calcular  $AB$ , sabiendo que  $A$  debe estar a la izquierda de  $B$ , primero debemos verificar que las matrices sean compatibles. Sólo es posible multiplicar dos matrices si el número de columnas de  $A$  es el mismo que el número de filas de  $B$ .

Por lo tanto, si  $A$  es una matriz de  $n \times m$  y  $B$  es una matriz de  $m \times k$ , entonces se puede encontrar el producto,  $AB$ , que será una nueva matriz de  $n \times k$ .

Para calcular el producto, realizamos una serie de multiplicaciones internas del producto entre los vectores fila de  $A$  y los vectores columna de  $B$ . La figura 5-2 ilustra el proceso para una matriz  $A$  de  $3 \times 3$  y una matriz  $B$  de  $3 \times 2$ .

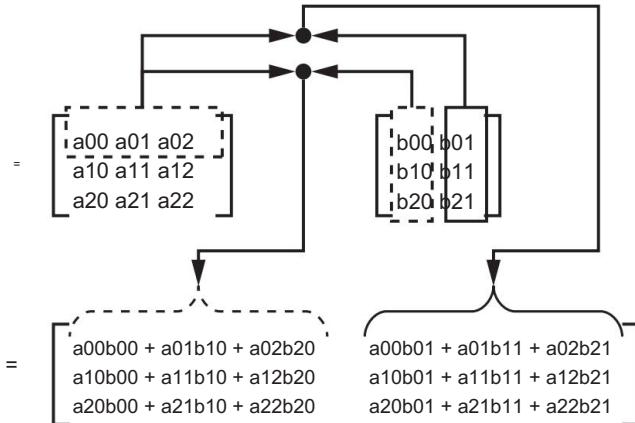


Figura 5-2: Multiplicar una matriz de  $3 \times 3$  por una matriz de  $3 \times 2$

En la Figura 5-2, la primera fila de la matriz de salida se encuentra calculando el producto interno de la primera fila de  $A$  con cada una de las columnas de  $B$ . Se muestra el primer elemento de la matriz de salida donde está la primera fila de  $A$  multiplicado por la primera columna de  $B$ . La primera fila restante de la matriz de salida se encuentra repitiendo el producto escalar de la primera fila de  $A$  por la columna restante de  $B$ .

Presentemos un ejemplo resuelto con números reales para las matrices en Figura 5-2:

$$AB = \begin{array}{ccc|cc} 1 & 2 & 3 & 11 & 22 \\ 4 & 5 & 6 & 33 & 44 \\ 7 & 8 & 9 & 55 & 66 \end{array}$$

$$\begin{aligned}
 &= (1)11 + (2)33 + (3)55 \quad (1)22 + (2)44 + (3)66 \\
 &\quad (4)11 + (5)33 + (6)55 \quad (4)22 + (5)44 + (6)66 \\
 &\quad (7)11 + (8)33 + (9)55 \quad (7)22 + (8)44 + (9)66
 \end{aligned}$$

$$\begin{aligned}
 &= 242 \quad 308 \\
 &\quad 539 \quad 704 \\
 &\quad 836 \quad 1100
 \end{aligned}$$

Observe que  $AB$  está definido, pero  $BA$  no, porque no podemos multiplicar una matriz de  $3 \times 2$  por una matriz de  $3 \times 3$ . El número de columnas de  $B$  debe ser igual al número de filas de  $A$ .

Otra forma de pensar en la multiplicación de matrices es considerando lo que implica cada uno de los elementos de la matriz de salida. Por ejemplo, si  $A$  es  $n \times m$  y  $B$  es  $m \times p$ , sabemos que el producto matricial existe como una matriz de  $n \times p$ ,  $C$ . Encontramos los elementos de salida calculando

$$c_{ij} = \sum_{k=0}^{m-1} a_{ik} b_{kj} \quad (5.9)$$

para  $i = 0, \dots, n - 1$  y  $j = 0, \dots, p - 1$ . En el ejemplo anterior, encontramos  $c_{21}$  sumando los productos  $a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$ , lo que se ajusta a la ecuación 5.9 con  $i = 2, j = 1$  y  $k = 0, 1, 2$ .

La ecuación 5.9 nos dice cómo encontrar un único elemento de matriz de salida. Si nosotros Si recorremos  $i$  y  $j$ , podemos encontrar la matriz de salida completa. Esto implica una implementación sencilla de la multiplicación de matrices:

---

```
def matrizmul(A,B):
    I,K = A.forma
    J = B.forma[1]
    C = np.zeros((I,J), dtype=A.dtype) para
    i en el rango(I): para
        j en el rango(J):
            para k en el
                rango(K): C[i,j] += A[i,k]*B[k,j]
    regresar C
```

---

Supondremos que los argumentos  $A$  y  $B$  son matrices compatibles. Establecimos el número de filas ( $I$ ) y columnas ( $J$ ) de la matriz de salida,  $C$ , y las usamos como límites de bucle para los elementos de  $C$ . Creamos la matriz de salida,  $C$ , y le damos el mismo tipo de datos que  $A$ . Luego comienza un triple bucle. El bucle sobre  $i$  cubre todas las filas de la salida. El siguiente bucle, sobre  $j$ , cubre las columnas de la fila actual, y el bucle más interno, sobre  $k$ , cubre la combinación de elementos de  $A$  y  $B$ , como en la ecuación 5.9. Cuando finalizan todos los ciclos, devolvemos el producto de la matriz,  $C$ .

La función `Matrixmul` funciona. Encuentra el producto de la matriz. Sin embargo, en términos de implementación, es bastante ingenuo. Existen algoritmos avanzados, al igual que muchas optimizaciones del enfoque ingenuo cuando se utiliza código compilado. Como veremos a continuación, NumPy admite la multiplicación de matrices y utiliza internamente bibliotecas de código compilado altamente optimizadas que superan con creces el rendimiento del código simple anterior.

Notación matricial para productos internos y

externos Ahora estamos en condiciones de comprender la notación matricial anterior para el producto interno,  $a \cdot b$ , y el producto externo,  $ab$  de dos vectores. En el primer caso, tenemos un vector de fila de  $1 \times n$ , debido a la transpuesta, y un vector de columna de  $n \times 1$ . El algoritmo dice que se forme el producto interno del vector fila.

y el vector columna para llegar a una matriz de salida que sea  $1 \times 1$ , es decir, un único número escalar. Observe que debe haber  $n$  componentes tanto en  $a$  como en  $b$ .

Para el producto exterior, tenemos un vector columna  $n \times 1$  a la izquierda y un vector fila  $1 \times m$  a la derecha. Por lo tanto, sabemos que la matriz de salida es  $n \times m$ . Si  $m = n$ , tendremos una matriz de salida que es  $n \times n$ . Una matriz con tantas filas como columnas es una matriz cuadrada. Estos tienen propiedades especiales, algunas de las cuales veremos en el Capítulo 6.

Para encontrar el producto exterior de dos vectores mediante multiplicación de matrices, multiplicamos cada elemento de las filas de  $a$  por cada una de las columnas de  $b$  como un vector fila,

$$\begin{aligned} ab &= \begin{matrix} a_0 \\ a_1 \\ a_2 \end{matrix} \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix} \\ &= \begin{matrix} a_0 b_0 & a_0 b_1 & a_0 b_2 \\ a_1 b_0 & a_1 b_1 & a_1 b_2 \\ a_2 b_0 & a_2 b_1 & a_2 b_2 \end{matrix} \end{aligned}$$

donde cada columna de  $b$  a, , un único número escalar, se transmite a lo largo de las filas de formando así cada producto posible entre los elementos de los dos vectores.

Hemos visto cómo realizar la multiplicación de matrices manualmente. Echemos un vistazo ahora a cómo NumPy admite la multiplicación de matrices.

#### Multiplicación de matrices en

NumPy NumPy proporciona dos funciones diferentes que podemos usar para la multiplicación de matrices. El primero, ya lo hemos visto, `np.dot`, aunque hasta ahora solo lo hemos usado para calcular productos internos de vectores. El segundo es `np.matmul`, al que también se llama cuando se utiliza el operador binario `@` disponible en Python 3.5 y posteriores. La multiplicación de matrices con cualquiera de las funciones funciona como esperamos. Sin embargo, NumPy a veces trata las matrices 1D de manera diferente a los vectores de fila o columna.

Podemos usar la forma para decidir si una matriz NumPy es una matriz 1D, un vector de fila o un vector de columna, como se muestra en el Listado 5-1:

---

```
>>> av = np.array([1,2,3])
>>> ar = np.array([[1,2,3]])
>>> ca = np.array([[1], [2],[3]])
>>> forma.av
(3,)
>>> forma.ar
(1, 3)
>>> forma.ac
(3, 1)
```

---

Listado 5-1: Vectores NumPy

Aquí vemos que una matriz 1D con tres elementos, av, tiene una forma diferente de un vector fila con tres componentes, ar, o un vector columna de tres componentes, ac. Sin embargo, cada una de estas matrices contiene los mismos tres números enteros: 1, 2 y 3.

Realicemos un experimento para ayudarnos a comprender cómo NumPy implementa la multiplicación de matrices. Probaremos np.dot, pero los resultados son los mismos si usamos np.matmul o el operador @ . Necesitamos una colección de vectores y matrices con los que trabajar. Luego aplicaremos combinaciones de ellos a np.dot y consideraremos el resultado, que muy bien puede ser un error si la operación no está definida para esa combinación de argumentos.

Creemos las matrices, vectores y matrices que necesitaremos:

---

```
a1 = np.array([1,2,3]) ar
= np.array([[1,2,3]]) ac =
np.array([[1],[2],[3]]) b1 =
np.array([1,2,3]) br =
np.array([[1,2,3]]) bc =
np.array([[1],[2],[3]])
A = np.matriz([[1,2,3],[4,5,6],[7,8,9]])
B = np.matriz([[9,8,7],[6,5,4],[3,2,1]])
```

---

La forma de los objetos debería ser discernible a partir de la definición, si tenemos en cuenta los resultados del Listado 5-1. También definiremos dos matrices de  $3 \times 3$ , A y B.

A continuación, definiremos una función auxiliar para ajustar la llamada a NumPy para que podamos detectar cualquier error:

---

```
def punto(a,b):
    intente: devolver np.punto(a,b)
    excepto:
        devolver "falla"
```

---

Esta función llama a np.dot y devuelve la palabra falla si la llamada no tiene éxito. La Tabla 5-1 muestra la salida de punto para las combinaciones dadas de las entradas definidas anteriormente.

La Tabla 5-1 ilustra cómo NumPy a veces trata las matrices 1D de manera diferente a los vectores de fila o columna. Vea la diferencia en la Tabla 5-1 para a1,A versus ar,A y A,ac. La salida de A,ac es la que esperaríamos ver matemáticamente, con el vector columna ac multiplicado a la izquierda por A.

¿ Existe alguna diferencia real entre np.dot y np.matmul? Sí, algo. Para matrices 1D y 2D, no hay diferencia. Sin embargo, existe una diferencia entre cómo cada función maneja matrices de más de dos dimensiones, aunque no trabajaremos con ellas aquí. Además, np.dot permite que uno de sus argumentos sea un escalar y multiplique cada elemento del otro argumento por él. Multiplicar por un escalar con np.matmul arroja un error.

Tabla 5-1: Resultados de la aplicación de dot o matmul a diferentes tipos de argumentos

Argumentos		Resultado de np.dot o np.matmul
a1,b1	14 (escalar)	
a1,br		falla
a1,bc		[14] (1 vector)
ar,b1		[14] (1 vector)
ar,br		falla
ar,bc		[14] (matriz $1 \times 1$ )
ac,b1		falla
		1 2 3 2 4 6 3 6 9
aae acondicionado, br		(producto exterior)
ca, antes de cristo		falla
A,a1		[14 32 50] (3 vectores)
ar		falla
		14
A,ac		32
		50
a1,A		[30 36 42] (3 vectores)
ara		[30 36 42] (matriz $1 \times 3$ )
CA,A		falla
		30 24 18
A,B		84 69 54
		138 114 90

## Producto Kronecker

La forma final de multiplicación de matrices que analizaremos es el producto de Kronecker o producto matricial directo de dos matrices. Al calcular la matriz producto, mezclamos elementos individuales de las matrices, multiplicándolos. Para el producto de Kronecker, multiplicamos los elementos de una matriz por una matriz completa para producir una matriz de salida que es más grande que las matrices de entrada. El producto Kronecker también es un lugar cómodo para presentar la idea de una matriz de bloques, o una matriz construida a partir de matrices más pequeñas (la bloques).

Por ejemplo, si tenemos tres matrices

$$\begin{array}{rccccc} & 1 & 2 & 3 & & 11 & 22 & & 111 \\ \text{Una} = & 4 & 5 & 6 & B = & 33 & 44 & C = & 222 \\ & 7 & 8 & 9 & & 55 & 66 & & 333 \end{array} \quad (5.10)$$

Podemos definir una matriz de bloques,  $M$ , de la siguiente manera.

$$\begin{array}{ccc}
 & 1 & 2 & 3 & 11 & 22 & 111 \\
 & 4 & 5 & 7 & 6 & 33 & 44 & 222 \\
 & 8 & & & 9 & 55 & 66 & 333 \\
 M = [ABC] & = & 11 & 22 & 111 & 1 \\
 & & 2 & 33 & 44 & 222 & 4 & 5 \\
 & & 3 & 6 & 55 & 66 & 333 & 7 & 8 & 9
 \end{array}$$

donde cada elemento de  $M$  es una matriz más pequeña apilada una encima de la otra.

Podemos definir más fácilmente el producto de Kronecker usando un ejemplo visual que involucra una matriz de bloques. El producto de Kronecker de  $A$  y  $B$ , típicamente escrito como  $A \otimes B$ , es

$$A \otimes B = \begin{matrix} a_{00}B & a_{01}B & a_{0,n-1}B & a_{11}B \\ a_{10}B & a_{11}B & & \\ & am-1,0B & am-1,1B & am-1,n-1B \end{matrix}$$

para  $A$ , una matriz  $m \times n$ . Esta es una matriz de bloques debido a  $B$ , por lo que, cuando se escribe completamente, el producto de Kronecker da como resultado una matriz mayor que  $A$  o  $B$ . Tenga en cuenta que, a diferencia de la multiplicación de matrices, el producto de Kronecker se define para matrices  $A$  y  $B$  de tamaño arbitrario. . Por ejemplo, utilizando  $A$  y  $B$  de la ecuación 5.10, el producto de Kronecker es

$$A \otimes B = \begin{matrix} (1)B & (2)B & (3)B \\ (4)B & (5)B & (6)B \\ (7)B & (8)B & (9)B \end{matrix} = \begin{matrix} 11 & 22 & 22 & 44 & 33 & 66 \\ 33 & 44 & 66 & 88 & 99 & 132 \\ 55 & 66 & 110 & 132 & 165 & 198 \\ 44 & 88 & 55 & 110 & 66 & 132 \\ 132 & 176 & 165 & 220 & 198 & 264 \\ 220 & 264 & 275 & 330 & 330 & 396 \\ 77 & 154 & 88 & 176 & 99 & 198 \\ 231 & 308 & 264 & 352 & 297 & 396 \\ 385 & 462 & 440 & 528 & 495 & 594 \end{matrix}$$

Observe arriba que usamos  $\otimes$  para el producto Kronecker. Esta es la convención, aunque a veces se abusa del símbolo  $\otimes$  y también se usa para otras cosas. Lo usamos para el producto exterior de dos vectores, por ejemplo. NumPy admite el producto Kronecker a través de `np.kron`.

## Resumen

En este capítulo, presentamos los objetos matemáticos utilizados en el aprendizaje profundo: escalares, vectores, matrices y tensores. Luego exploramos la aritmética con tensores, en particular con vectores y matrices. Vimos cómo realizar operaciones sobre estos objetos, tanto matemáticamente como en código a través de NumPy.

Sin embargo, nuestra exploración del álgebra lineal no está completa. En el próximo capítulo, profundizaremos en las matrices y sus propiedades para analizar solo algunas de las cosas importantes que podemos hacer con ellas o saber sobre ellas.

# 6

## MORELINEARALGEBRA



En este capítulo, continuaremos nuestra exploración de conceptos de álgebra lineal. Algunos de estos conceptos sólo están relacionados tangencialmente con el aprendizaje profundo, pero son el tipo de matemáticas que eventualmente encontrarás. Piense en este capítulo como si se supusiera un conocimiento previo.

Especificamente, aprenderemos más sobre las propiedades y operaciones con matrices cuadradas, presentando términos que encontrará en la literatura sobre aprendizaje profundo. Después de eso, presentaré las ideas detrás de los valores propios y vectores propios de una matriz cuadrada y cómo encontrarlos. A continuación, exploraremos las normas vectoriales y otras formas de medir la distancia que a menudo se encuentran en el aprendizaje profundo. En ese punto, presentaré el importante concepto de matriz de covarianza.

Concluiremos el capítulo demostrando el análisis de componentes principales (PCA) y la descomposición de valores singulares (SVD). Estos enfoques utilizados con frecuencia dependen en gran medida de los conceptos y operadores introducidos a lo largo del capítulo. Veremos qué es PCA, cómo hacerlo y qué nos puede aportar desde la perspectiva del aprendizaje automático. De manera similar, trabajaremos con SVD y veremos cómo podemos usarlo para implementar PCA y también para calcular la pseudoinversa de una matriz rectangular.

## Matrices cuadradas

Las matrices cuadradas ocupan un lugar especial en el mundo del álgebra lineal. Explorémoslos con más detalle. Los términos utilizados aquí aparecerán a menudo en el aprendizaje profundo y otras áreas.

¿Por qué matrices cuadradas?

Si multiplicamos una matriz por un vector columna, obtendremos otro vector columna como salida:

$$\begin{matrix} & & 11 \\ & & 12 \\ [8] & 3 & 4 & 5 & 6 & 7 & 13 \\ & & & & & & 14 \end{matrix} = [130 \ 330]$$

Interpretada geométricamente, la matriz  $2 \times 4$  ha mapeado la columna  $4 \times 1$ . El vector umn, un punto en  $\mathbb{R}^4$ , a un nuevo punto en  $\mathbb{R}^2$  mapeo es lineal porque los valores de los puntos solo se multiplican por los elementos de la matriz de  $2 \times 4$ ; no existen operaciones no lineales, como elevar las componentes del vector a una potencia, por ejemplo.

Visto de esta manera, podemos usar una matriz para transformar puntos entre espacios. Si la matriz es cuadrada, digamos,  $n \times n$ , el mapeo es desde  $\mathbb{R}^n$  hasta  $\mathbb{R}^n$ . Por ejemplo, considere

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \begin{matrix} 11 \\ 12 \\ 13 \end{matrix} = \begin{matrix} 74 \\ 182 \\ 209 \end{matrix}$$

donde el punto  $(11, 12, 13)$  se asigna al punto  $(74, 182, 209)$ , ambos en  $\mathbb{R}^3$ .

El uso de una matriz para asignar puntos de un espacio a otro hace posible rotar un conjunto de puntos alrededor de un eje mediante el uso de una matriz de rotación. Para rotaciones simples, podemos definir matrices en 2D,

$$R(\theta) = [\cos \theta \ -\sin \theta \ \sin \theta \ \cos \theta] \quad (6.1)$$

y en 3D,

$$\begin{matrix} 1 & 0 & 0 \\ 0 & \text{porque} & \text{pecado} \\ 0 & \text{pecado} & \text{porque} \end{matrix} \begin{matrix} 0 \\ 0 \\ x \end{matrix} \begin{matrix} \text{porque} \\ \text{pecado} \\ \text{porque} \end{matrix} \begin{matrix} 0 \\ 1 \\ y \end{matrix} \begin{matrix} \text{porque} - \text{pecado} \\ \text{pecado} \\ \text{porque} \end{matrix} \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \begin{matrix} 0 \\ 0 \\ z \end{matrix}$$

Las rotaciones son en ángulo, y para 3D, alrededor del eje x, y o z, como lo indica el subíndice.

Usando una matriz, podemos crear una transformación afín. Una transformación afín mapea un conjunto de puntos en otro conjunto de puntos de modo que los puntos en un

Línea en el espacio original todavía están en una línea en el espacio mapeado. La transformación es

$$y = Ax + b$$

La transformación afín combina una transformación matricial, A, con una traducción, b, para asignar un vector, x, a un nuevo vector, y. Podemos combinar esta operación en una única multiplicación de matrices colocando A en la esquina superior izquierda de la matriz y agregando b como una nueva columna a la derecha. Una fila de ceros en la parte inferior con un solo 1 en la columna más a la derecha completa la matriz de transformación aumentada. Para una matriz de transformación afín

$$\begin{bmatrix} A & B \end{bmatrix}$$

y vector de traducción

$$\begin{bmatrix} ij \end{bmatrix}$$

obtenemos

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} abi & x \\ cdj & y \\ 0 & 1 \end{bmatrix}$$

Esta forma asigna un punto, (x,y), a un nuevo punto, (x''), y

Esta maniobra es idéntica al truco de sesgo que a veces se utiliza al implementar una red neuronal para enterrar el sesgo en una matriz de peso aumentada al incluir una entrada de vector de características adicional establecida en 1. De hecho, podemos ver una red neuronal de avance como una serie de transformaciones afines, donde la matriz de transformación es la matriz de peso entre las capas y el vector de sesgo proporciona la traducción. La función de activación en cada capa altera la relación lineal entre las capas. Es esta no linealidad la que permite a la red aprender una nueva forma de mapear entradas para que la salida final refleje la relación funcional para la cual la red está diseñada para aprender.

Entonces, usamos matrices cuadradas para mapear puntos de un espacio nuevamente al mismo espacio, por ejemplo, para rotarlos alrededor de un eje. Veamos ahora algunas propiedades especiales de las matrices cuadradas.

### Transposición, rastreo y potencias

El capítulo 5 nos mostró la transposición de vectores para movernos entre vectores de columna y fila. La operación de transposición no se limita a vectores. Funciona para cualquier matriz volteando las filas y columnas a lo largo de la diagonal principal. Por ejemplo,

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} = \begin{array}{ccc} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{array} , [abcde\bar{f}gh] \quad \begin{array}{l} ae \\ novio \\ cg \\ DH \end{array}$$

La transpuesta se forma invirtiendo los índices de los elementos de la matriz:

$$a_{ji} \leftarrow a_{ij}, i = 0, 1, \dots, \text{norte}-1, j = 0, 1, \dots, \text{metro}-1$$

Esto cambia una matriz  $n \times m$  en una matriz  $m \times n$ . Observe que el orden de una matriz cuadrada sigue siendo el mismo bajo la operación de transposición y los valores en la diagonal principal no cambian.

En NumPy, puedes llamar al método de transposición en una matriz, pero la transposición es tan común que también existe una notación abreviada (.T) . Por ejemplo,

---

```
>>> importar numpy
como np >>> a = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>>

imprimir(a)
[[1 2 3] [4 5 6] [7 8 9]] >>>
```

---

```
imprimir(a.transpose()) [[1 4 7] [2 5 8] [3 6 9]] >>> imprimir(aT ) [[1 4 7] [2 5 8] [3 6 9]]
```

---

La traza es otra operación común aplicada a matrices cuadradas:

$$\text{tr}A = \sum_{i=0}^{\text{norte}-1} a_{ii}$$

Como operador, la traza tiene ciertas propiedades. Por ejemplo, es lineal:

$$\text{tr}(A + B) = \text{tr}A + \text{tr}B$$

También es cierto que  $\text{tr}(A) = \text{tr}(A^t)$  y  $\text{tr}(AB) = \text{tr}(BA)$ .

NumPy usa np.trace para calcular rápidamente la traza de una matriz y np.diag para devolver los elementos diagonales de una matriz como una matriz 1D,

$$(a_{00}, a_{11}, \dots, a_{n-1,n-1})$$

para una matriz  $n \times n$  o  $n \times m$ .

No es necesario que una matriz sea cuadrada para que NumPy devuelva los elementos a lo largo de su diagonal. Y aunque matemáticamente la traza generalmente solo se aplica a matrices cuadradas, NumPy calculará la traza de cualquier matriz, devolviendo la suma de los elementos diagonales:

---

```
>>> b = np.array([[1,2,3,4],[5,6,7,8]]) >>>
imprimir(b)
```

```
[1 2 3 4] [5
6 7 8]] >>>
imprimir(np.diag(b)) [1 6]
>>>
imprimir(np.trace(b))
7
```

---

Por último, puedes multiplicar una matriz cuadrada por sí misma, lo que implica que puedes elevar una matriz cuadrada a una potencia entera, n, multiplicándose n veces.

Tenga en cuenta que esto no es lo mismo que elevar los elementos de la matriz a una potencia. Por ejemplo,

$$A^2 = AA = [1 2 3 4][1 6] = [7 15 0 22] \neq [1 4 9 16]$$

La potencia matricial sigue las mismas reglas que elevar cualquier número a una potencia:

$$A^n A^m = A^{n+m}$$

$$(A^n)^m = A^{nm}$$

NumPy  $A^n$  (enteros positivos) y donde A es una matriz cuadrada. para n, m proporciona una función para calcular la potencia de una matriz cuadrada de manera más eficiente que las llamadas repetidas a np.dot:

```
>>> de numpy.linalg importar potencia_matriz >>> a =
np.array([[1,2],[3,4]]) >>>
imprimir(potencia_matriz(a,2)) [[ 7 10]
[15 22]]
>>>
print(matrix_power(a,10)) [[ 4783807
6972050] [10458075
15241882]]
```

---

Ahora consideremos algunas matrices cuadradas especiales que encontrará de vez en cuando.

### Matrices cuadradas especiales

Muchas matrices cuadradas (y no cuadradas) han recibido nombres especiales. Algunas son bastante obvias, como las matrices que son todas cero o uno, que se llaman matrices de ceros y matrices de unos, respectivamente. NumPy los utiliza ampliamente:

```
>>> imprimir(np.zeros((3,5))) [[0. 0.
0. 0. 0.] [0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0.]] >>>
print(np.ones(3,3)) [[1. 1.
1.] [1. 1. 1.]
[1. 1. 1.]]
```

---

Tenga en cuenta que puede encontrar una matriz de cualquier valor constante, c, multiplicando la matriz de las unidades por c.

Observe arriba que NumPy utiliza por defecto matrices de números de punto flotante de 64 bits correspondientes a un tipo de doble en lenguaje C. Consulte la Tabla 1-1 en la página 6 para obtener una lista de posibles tipos de datos numéricos. Puede especificar el tipo de datos deseado con la palabra clave `dtype`. En matemáticas puras, no nos importan mucho los tipos de datos, pero para trabajar en aprendizaje profundo, es necesario prestar atención para evitar definir matrices que consuman mucha más memoria de la necesaria. Muchos modelos de aprendizaje profundo están satisfechos con matrices de flotantes de 32 bits, que utilizan la mitad de memoria por elemento que el valor predeterminado de NumPy. Además, muchos kits de herramientas utilizan tipos de datos nuevos o raramente utilizados, como flotantes de 16 bits, para permitir un uso aún mejor de la memoria. NumPy admite flotantes de 16 bits al especificar `float16` como tipo d.

### La matriz de identidad

Con diferencia, la matriz especial más importante es la matriz de identidad. Esta es una matriz cuadrada con todos los unos en la diagonal:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix} \quad \text{yo = } \begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \quad (6.2)$$

La matriz identidad actúa como el número 1 al multiplicar una matriz. Por lo tanto,

$$IA = IA = A$$

para una matriz cuadrada A de  $n \times n$  y una matriz identidad I de  $n \times n$ . Cuando sea necesario, agregaremos un subíndice para indicar el orden de la matriz identidad, por ejemplo,  $I_n$ .

NumPy usa `np.identity` o `np.eye` para generar matrices de identidad de un tamaño determinado:

---

```
>>> a = np.array([[1,2],[3,4]]) >>> i =
np.identidad(2) >>> print(i)
[[1. 0.] [0. 1.]]
>>>
imprimir(a
@ i)
```

---

[[1. 2.]  
[3. 4.]]

---

Mire atentamente el ejemplo anterior. Matemáticamente, dijimos que la multiplicación de una matriz cuadrada por la matriz identidad del mismo orden devuelve la matriz. NumPy, sin embargo, hizo algo que quizás no queramos. La matriz `a` se definió con elementos enteros, por lo que tiene un tipo de datos de `int64`, el valor predeterminado de NumPy para números enteros. Sin embargo, dado que no proporcionamos explícitamente a `np.identity` un tipo de datos, NumPy adoptó de forma predeterminada un flotante de 64 bits. Por lo tanto, la multiplicación de matrices (@) entre `a` y `devolvío` una versión de punto flotante de `a`. Este cambio sutil del tipo de datos puede ser importante para cálculos posteriores, por lo que, nuevamente, debemos prestar atención a los tipos de datos cuando usamos NumPy.

No importa si usas `np.identity` o `np.eye`. De hecho, internamente, `np.identity` es solo un contenedor para `np.eye`.

### Matrices

**triangulares** De vez en cuando oirás hablar de matrices triangulares. Hay dos tipos: superior e inferior. Como puede intuir por el nombre, una matriz triangular superior es aquella con elementos distintos de cero en la parte que está sobre o sobre la diagonal principal, mientras que una matriz triangular inferior solo tiene elementos sobre o debajo de la diagonal principal. Por ejemplo,

$$\begin{matrix} & 1 & 2 & 3 & 4 \\ tu = & 0 & 5 & 6 & 7 \\ & 0 & 0 & 8 & 9 \\ & 0 & 0 & 0 & 10 \end{matrix}$$

es una matriz triangular superior, mientras que

$$\begin{matrix} & 1 & 0 & 0 & 0 \\ I = & 2 & 3 & 0 & 0 \\ & 4 & 5 & 6 & 0 \\ & 7 & 8 & 9 & 10 \end{matrix}$$

es una matriz triangular inferior. Una matriz que tiene elementos sólo en la diagonal principal es, como era de esperar, una matriz diagonal.

NumPy tiene dos funciones, `np.triu` y `np.tril`, para devolver la parte superior o parte triangular inferior de la matriz dada, respectivamente. Entonces,

---

```
>>> a = np.arange(16).reshape((4,4))
>>> print(a)
[[ 0  1  2  3] [ 4
 5  6  7] [ 8  9
10 11] [12 13
14 15]] >>>
imprimir(np.triu(a))
[[ 0  1  2  3] [ 0
 5  6  7] [ 0  0
10 11]]
```

```
[ 0 0 0 15]
>>> print(np.tril(a))
[[ 0 0 0 0] [ 4
 5 0 0] [ 8 9
 10 0] [12 13
 14 15]]
```

---

No utilizamos con frecuencia matrices triangulares en el aprendizaje profundo, pero sí usarlos en álgebra lineal, en parte para calcular determinantes, a los que ahora gira.

## Determinantes

Podemos pensar en el determinante de una matriz cuadrada,  $n \times n$ , como una función que asigna matrices cuadradas a un escalar. El uso principal del determinante en el aprendizaje profundo es calcular los valores propios de una matriz. Veremos lo que eso significa más adelante en este capítulo, pero por ahora piense en los valores propios como valores escalares especiales asociados con una matriz. El determinante también nos dice algo sobre si una matriz tiene inversa o no, como también veremos a continuación.

Notacionalmente, escribimos el determinante de una matriz con barras verticales. Por ejemplo, si  $A$  es una matriz de  $3 \times 3$ , escribimos el determinante como

$$\det(A) = \begin{vmatrix} a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \end{vmatrix}$$

donde afirmamos explícitamente que el valor del determinante es un escalar (elemento de  $\mathbb{R}$ ). Todas las matrices cuadradas tienen un determinante. Por ahora, consideraremos algunas de las propiedades del determinante:

1. Si alguna fila o columna de  $A$  es cero, entonces  $\det(A) = 0$ .
2. Si dos filas cualesquiera de  $A$  son idénticas, entonces  $\det(A) = 0$ .
3. Si  $A$  es una matriz triangular superior o inferior, entonces  $\det(A) = \prod_{i=1}^n a_{ii}$ .
4.  $A$  es una matriz diagonal, entonces  $\det(A) = \prod_{i=1}^n a_{ii}$ . El determinante de la matriz identidad, independientemente de tamaño, es 1.
5. El determinante de un producto de matrices es el producto de la determinantes,  $\det(AB) = \det(A)\det(B)$ .
6.  $\det(A^\top) = \det(A)$
7.  $\det(A^{-1}) = \frac{1}{\det(A)}$
8.  $\det(A^\top) = \det(A)$

La propiedad 7 indica que la operación de transposición no cambia el valor de un determinante. La propiedad 8 es consecuencia de la propiedad 6.

Tenemos varias formas de calcular el determinante de una matriz cuadrada. Aquí examinaremos sólo una forma, que implica el uso de una fórmula recursiva. Todas las fórmulas recursivas se aplican a sí mismas, del mismo modo que las funciones recursivas en el código se llaman a sí mismas. La idea general es que cada recursión funciona en un

versión más simple del problema, que se puede combinar para devolver la solución al problema más grande.

Por ejemplo, podemos calcular el factorial de un número entero,

$$!norte = norte(norte - 1)(norte - 2)(norte - 3) \dots 1$$

recursivamente si notamos lo siguiente:

$$!norte = norte \times (norte - 1)!$$

$$0! = 1$$

La primera afirmación dice que el factorial de  $n$  es  $n$  veces el factorial de  $(n - 1)$ . El segundo enunciado dice que el factorial de cero es uno. La recursión es la primera declaración, pero esta recursión nunca terminará sin alguna condición que devuelva un valor. Ese es el punto de la segunda afirmación, el caso base: dice que la recursividad termina cuando llegamos a cero.

Esto podría quedar más claro en el código. Podemos definir el factorial así:

---

```
def factorial(n):
    si (n == 0):
        devuelve 1
    devolver n*factorial(n-1)
```

---

Observe que factorial se llama a sí mismo con el argumento menos uno, a menos que el argumento sea cero, en cuyo caso devuelve uno inmediatamente. El código funciona gracias a la pila de llamadas de Python. La pila de llamadas realiza un seguimiento de todos los cálculos de  $n * factorial(n-1)$ . Cuando encontramos el caso base, se realizan todas las multiplicaciones pendientes y devolvemos el valor final.

Entonces, para calcular determinantes de forma recursiva, necesitamos una declaración de recursividad, algo que defina el determinante de una matriz en términos de determinantes más simples. También necesitamos un caso base que nos dé un valor definitivo. Para los determinantes, el caso base es cuando llegamos a una matriz de  $1 \times 1$ . Para cualquier matriz  $1 \times 1$ , A, tenemos

$$\det(A) = a_{00}$$

lo que significa que el determinante de una matriz de  $1 \times 1$  es el valor único que contiene.

Nuestro plan es calcular el determinante dividiendo el cálculo en determinantes sucesivamente más simples hasta llegar al caso base anterior. Para hacer esto, necesitamos una declaración que implique recursividad. Sin embargo, necesitamos definir algunas cosas antes de poder hacer la declaración. Primero, necesitamos definir el menor de una matriz. La  $(i, j)$ -menor de una matriz, A, es la matriz que queda después de eliminar la  $i$ -ésima fila y la  $j$ -ésima columna de A. Denotaremos una matriz menor por  $A_{ij}$ . Por ejemplo, dado

$$\begin{array}{ccc} 9 & 8 & 7 \\ \text{Una} = & 6 & 5 & 4 \\ & 3 & 2 & 1 \end{array}$$

entonces

$$A_{11} = \begin{array}{r} 9 \ 8 \ 7 \\ \hline 6 \ 5 \ 4 \\ \hline 3 \ 2 \ 1 \end{array} = [9]3$$

donde el menor,  $A_{11}$ , se encuentra eliminando la fila 1 y la columna 1 para dejar solo los valores subrayados.

En segundo lugar, necesitamos definir el cofactor,  $C_{ij}$ , del menor,  $A_{ij}$ . Esto es donde aparece nuestra declaración recursiva. La definición es

$$C_{ij} = (-1)^{i+j+2} \det(A_{ij})$$

El cofactor depende del determinante del menor. Observa el exponente de  $-1$ , escrito como  $i + j + 2$ . Si miras la mayoría de los libros de matemáticas, verás el exponente como  $i + j$ . Hemos tomado una decisión consciente de definir matrices con índices de base cero para que las matemáticas y la implementación en el código coincidan sin desviarse por uno. Aquí hay un lugar donde esa elección nos obliga a ser menos elegantes que los textos de matemáticas. Debido a que nuestros índices están "desviados" en uno, debemos agregar ese nuevamente al exponente del cofactor para que el patrón de valores positivos y negativos que utiliza el cofactor sea correcto. Esto significa sumar uno a cada una de las variables en el exponente:  $i \rightarrow i + 1$  y  $j \rightarrow j + 1$ . Esto hace que el exponente  $i + j \rightarrow (i + 1) + (j + 1) = i + j + 2$ .

Ahora estamos listos para nuestra definición recursiva completa del determinante de  $A$  mediante el uso de expansión de cofactor. Resulta que sumar el producto de los valores de la matriz y los cofactores asociados para cualquier fila o columna de una matriz cuadrada nos dará el determinante. Entonces, usaremos la primera fila de la matriz y calcularemos el determinante como

$$\det(A) = a_0 \sum_{j=0}^{n-1} C_{0j} \quad (6.3)$$

Quizás se pregunte: ¿Dónde está la recursividad en la Ecuación 6.3? Aparece en el determinante del menor. Si  $A$  es una matriz  $n \times n$ , la menor,  $A_{ij}$ , es una matriz  $(n - 1) \times (n - 1)$ . Por lo tanto, para calcular los cofactores para encontrar el determinante de una matriz  $n \times n$ , necesitamos saber cómo encontrar el determinante de una matriz  $(n - 1) \times (n - 1)$ . Sin embargo, podemos usar la expansión de cofactores para encontrar el determinante  $(n - 1) \times (n - 1)$ , lo que implica encontrar el determinante de una matriz  $(n - 2) \times (n - 2)$ . Este proceso continúa hasta llegar a una matriz de  $1 \times 1$ . Ya sabemos que el determinante de una matriz de  $1 \times 1$  es el valor único que contiene.

Sigamos este proceso para una matriz de  $2 \times 2$ :

$$A = [abcd]$$

Usando la expansión de cofactores, obtenemos

$$\det(A) = a00C00 + a01C01$$

$$= aC00 + bC01$$

$$= a(-1)^{0+0+2}\det(A00) + b(-1)^{0+1+2}\det(A01)$$

= anuncio - antes de Cristo

que es la fórmula para el determinante de una matriz de  $2 \times 2$ . Los menores de una matriz de  $2 \times 2$  son matrices de  $1 \times 1$ , y cada una de ellas devuelve d o c en este caso.

En NumPy, calculamos determinantes con `np.linalg.det`. Por ejemplo,

---

```
>>> a = np.array([[1,2],[3,4]]) >>>
imprimir(a)
[[1
2] [3
4]] >>>
np.linalg.det(a )
-2.0000000000000004
>>> 1*4 - 2*3
-2
```

---

La última línea de código utiliza la fórmula para una matriz de  $2 \times 2$  que derivamos anteriormente con fines de comparación. Internamente, NumPy no utiliza la expansión recursiva del cofactor para calcular el determinante. En cambio, factoriza la matriz en el producto de tres matrices: (1) una matriz de permutación, que parece una matriz de identidad revuelta con solo una en cada fila y columna, (2) una matriz triangular inferior y (3) una matriz triangular superior. El determinante de la matriz de permutación es  $+1$  o  $-1$ . El determinante de una matriz triangular es el producto de los elementos diagonales, mientras que el determinante de un producto de matrices es el producto de los determinantes por matriz.

Podemos usar determinantes para determinar si una matriz tiene inversa.

Vayamos allí ahora.

## Inversos

La ecuación 6.2 define la matriz identidad. Dijimos que esta matriz actúa como el número 1, por lo que cuando multiplica una matriz cuadrada, se devuelve la misma matriz cuadrada. Al multiplicar escalares, sabemos que para cualquier número,  $x \neq 0$ , existe otro número, llámelo  $y$ , tal que  $xy = 1$ . Este número es el inverso multiplicativo de  $x$ . Además, sabemos exactamente qué es  $y$ ; es  $1/x = x^{-1}$ .

Entonces, por analogía, podríamos preguntarnos si, dado que tenemos una matriz identidad que actúa como el número 1, existe otra matriz cuadrada, llamémosla  $A^{-1}$ , para una matriz cuadrada dada,  $A$ , tal que

$$AA^{-1} = A^{-1}A = \text{Yo}$$

si un  $A^{-1}$  existe, se conoce como matriz inversa de  $A$ , y se dice que  $A$  es invertible. Para los números reales, todos los números excepto el cero tienen inverso. Para las matrices, no es tan sencillo. Muchas matrices cuadradas no tienen inversos. Para comprobar si  $A$  tiene inversa, usamos el determinante:  $\det(A) = 0$  nos dice que  $A$  no tiene inversa. Además, si  $A^{-1}$  existe, entonces

$$\det(A^{-1}) = \frac{1}{\det(A)}$$

Tenga en cuenta también que  $(A^{-1})^{-1} = A$ , como es el caso de los números reales. Otra propiedad útil de las inversas es

$$(AB)^{-1} = B^{-1}A^{-1}$$

donde es importante el orden del producto en el lado derecho. Finalmente, observe que la inversa de una matriz diagonal es simplemente el recíproco de los elementos diagonales:

$$\text{Una} = \begin{matrix} \text{un } 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{matrix} \rightarrow \text{UN}^{-1} = \begin{matrix} \text{un }^{-1} & 0 \\ 0 & 0 & \text{segundo } 0 \\ 0 & 0 & -1 \end{matrix}$$

Es posible calcular la inversa a mano usando operaciones de fila, que hemos ignorado convenientemente aquí porque rara vez se usan en el aprendizaje profundo. Las técnicas de expansión de cofactores también pueden calcular el inverso, pero para ahorrar tiempo, no daremos más detalles sobre el proceso aquí. Lo que es importante para nosotros es saber que las matrices cuadradas a menudo tienen una inversa y que podemos calcular inversas con NumPy a través de `np.linalg.inv`.

Si una matriz no es invertible, se dice que es singular. Por tanto, el determinante de una matriz singular es cero. Si una matriz tiene inversa, es una matriz no singular o no degenerada.

En NumPy, usamos `np.linalg.inv` para calcular la inversa de una matriz cuadrada. Por ejemplo,

---

```
>>> a = np.array([[1,2,1],[2,1,2],[1,2,2]]) >>>
imprimir(a)
[[1 2 1]
 [2 1 2]
 [1 2 2]]
>>> b = np.linalg.inv(a)
>>>
imprimir(b) [[ 0,66666667 0,66666667 -1. ]]
```

```
[ 0,66666667 -0,33333333 0. ] [-1. 1. ]
>>> imprimir(a @ b) [[1. 0. 0.] [0. 1. 0.]
[0. 0. 1.]] >>
imprimir(b @
a) [[1. 0. 0.]
[0. 1. 0.] [0. 0.
1.]]
```

---

Observe que la inversa (b) funciona como esperamos y da la matriz identidad al multiplicar a por la izquierda o por la derecha.

### Matrices simétricas, ortogonales y unitarias

Si para una matriz cuadrada, A, tenemos

$$A = U_n$$

entonces se dice que A es una matriz simétrica. Por ejemplo,

$$\text{Una} = \begin{matrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 6 & 7 \\ 3 & 6 & 8 & 9 \\ 4 & 7 & 9 & 1 \end{matrix}$$

es una matriz simétrica, ya que  $A^T = A$ .

Observe que las matrices diagonales son simétricas y el producto de dos matrices simétricas es comutativo:  $AB = BA$ . La inversa de una matriz simétrica, si existe, también es una matriz simétrica.

Si lo siguiente es cierto,

$$AA = A \quad A = Y_o$$

entonces A es una matriz ortogonal. Si A es una matriz ortogonal, entonces

$$A^{-1} = U_n$$

y como un resultado,

$$\det(A) = \pm 1$$

Si se permite que los valores en la matriz sean complejos, lo cual no sucede a menudo en el aprendizaje profundo, y

$$U^T U = UU^T = Y_o$$

entonces  $U$  es una matriz unitaria      siendo la transpuesta conjugada de  $U$ . La con-  
con  $U$  yugate transpuesta es la transpuesta de matriz ordinaria seguida de la  
operación conjugada compleja para cambiar  $i = \sqrt{-1}$  a  $-i$ . Entonces, podríamos tener

$$U = [1 -3i 2i] \quad 3i 3i + 2i \quad = [1 3 3i] \quad 2i 2 + 3i$$

A veces, especialmente en física, la transpuesta conjugada se llama adjunta hermitiana y se denota como  $A^\dagger$ . Si una matriz es igual a su transpuesta conjugada, se llama matriz hermitiana. Observe que las matrices simétricas reales también son matrices hermitianas porque la transpuesta conjugada es la misma que la transpuesta ordinaria cuando los valores son números reales. Por lo tanto, es posible que te encuentres con el término hermitiano en lugar de simétrico cuando te refieres a matrices con elementos reales.

Definitividad de una matriz simétrica Vimos

al principio de esta sección que una matriz cuadrada de  $n \times n$  asigna un vector en  $\mathbb{R}^n$  a otro vector en  $\mathbb{R}^n$ . Consideremos ahora una matriz simétrica  $n \times n$ ,  $B$ , con elementos de valor real. Podemos caracterizar esta matriz por cómo asigna vectores utilizando el producto interno entre el vector asignado y el vector original. Específicamente, si  $x$  es un vector columna ( $n \times 1$ ), entonces  $Bx$  también es un vector columna  $n \times 1$ . Por lo tanto, el producto interno de este vector y el vector original,  $x$ , es  $x^T Bx$ , un escalar.

Si lo siguiente es cierto:

$$x^T Bx > 0, \quad x \neq 0$$

entonces se dice que  $B$  es definido positivo. Aquí, el 0 en negrita es el vector columna  $n \times 1$  de todos los ceros, y  $\forall$  es la notación matemática que significa "para todos".

De manera similar, si

$$x^T Bx < 0, \quad x \neq 0$$

entonces  $B$  es definido negativo. Relajar la relación interna del producto y el requisito distinto de cero en  $x$  da dos casos adicionales. Si

$$x^T Bx \geq 0, \quad x \in \mathbb{R}^{n \times 1}$$

entonces se dice que  $B$  es semidefinido positivo, y

$$x^T Bx \leq 0, \quad x \in \mathbb{R}^{n \times 1}$$

hace que  $B$  sea una matriz semidefinida negativa. Finalmente, una matriz simétrica cuadrada real que no es semidefinida positiva ni negativa se llama matriz indefinida. La precisión de una matriz nos dice algo sobre los valores propios, sobre los cuales aprenderemos más en la siguiente sección. Si una matriz simétrica es

definida positiva, entonces todos sus valores propios son positivos. De manera similar, una matriz definida negativa simétrica tiene todos valores propios negativos. Las matrices simétricas semidefinidas positivas y negativas tienen valores propios que son todos positivos o cero o todos negativos o cero, respectivamente.

Pasemos ahora de hablar de tipos de matrices a descubrir la importancia de los vectores propios y los valores propios, propiedades clave de una matriz que utilizamos con frecuencia en el aprendizaje profundo.

## Vectores propios y valores propios

Aprendimos anteriormente que una matriz cuadrada asigna un vector a otro vector en el mismo espacio dimensional,  $v$  vectores, si  $A' = Av$ , donde ambos  $v'$  y  $v$  son  $n$ -dimensionales es una matriz  $n \times n$ .

Considere esta ecuación,

$$Av = v \quad (6.4)$$

para alguna matriz cuadrada,  $A$ , donde  $\lambda$  es un valor escalar y  $v$  es un vector de columna distinto de cero.

La ecuación 6.4 dice que  $A$  convierte el vector  $v$  en un múltiplo escalar de sí mismo. Llamamos a  $v$  un vector propio de  $A$  con valor propio. El prefijo eigen proviene del alemán y a menudo se traduce como "propio", "característico" o incluso "propio". Pensando geométricamente, la ecuación 6.4 dice que la acción de  $A$  sobre sus vectores propios en  $n$  es reducir o expandir el vector sin cambiar su dirección. Tenga en cuenta que, si bien  $v$  es distinto de cero, es posible que sea cero.

¿Cómo se relaciona la ecuación 6.4 con la matriz identidad,  $I$ ? Por definición, la matriz identidad mapea un vector dentro de sí mismo sin escalarlo. Por tanto, la matriz identidad tiene un número infinito de vectores propios, y todos ellos tienen un valor propio de 1, ya que, para cualquier  $x$ ,  $Ix = x$ . Por lo tanto, el mismo valor propio puede aplicarse a más de un vector propio.

Recuerde que la ecuación 6.1 define una matriz de rotación en el espacio 2D para algunos ángulo dado, . Esta matriz no tiene vectores propios porque, para cualquier vector distinto de cero, rota el vector , por lo que nunca puede mapear un vector nuevamente en su dirección original. Por tanto, no todas las matrices tienen vectores propios.

### Encontrar valores propios y vectores propios

Para encontrar los valores propios de una matriz, si los hay, volvemos a la ecuación 6.4 y la reescribimos:

$$Av - v = Av - Iv = (A - I)v = 0 \quad (6.5)$$

Podemos insertar la matriz identidad,  $I$ , entre  $v$  y  $v$  porque  $Iv = v$ . Por lo tanto, para encontrar los valores propios de  $A$ , necesitamos encontrar valores de que hagan que la matriz  $A - I$  asigne un vector distinto de cero,  $v$ , a el vector cero. La ecuación 6.5 sólo tiene soluciones distintas al vector cero si el determinante de  $A - I$  es cero.

Lo anterior nos da una manera de encontrar los valores propios. Por ejemplo, considere cómo se ve  $A - I$  para una matriz de  $2 \times 2$ :

$$A - I = [ab \quad \text{disco compacto}] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= [ab \quad cd] - \begin{bmatrix} a & 0 \\ 0 & c \end{bmatrix}$$

$$\begin{bmatrix} b & -c \\ -d & a \end{bmatrix}$$

Aprendimos anteriormente que el determinante de una matriz de  $2 \times 2$  tiene una simple forma; por lo tanto, el determinante de la matriz anterior es

$$\det(A - I) = (a - )(d - ) - \text{antes de Cristo}$$

Esta ecuación es un polinomio de segundo grado en  $\lambda$ . Como necesitamos que el determinante sea cero, ponemos este polinomio en cero y encontramos las raíces. El las raíces son los valores propios de  $A$ . El polinomio que encuentra este proceso es llama polinomio característico, y la ecuación 6.5 es la ecuación característica. Observe arriba que el polinomio característico es un polinomio de segundo grado. En general, el polinomio característico de una matriz  $n \times n$  es de grado  $n$ , por lo que una matriz tiene como máximo  $n$  valores propios distintos, ya que un enésimo grado El polinomio tiene como máximo  $n$  raíces.

Una vez que conocemos las raíces del polinomio característico, podemos ir Volviendo a la ecuación 6.5, sustituya cada raíz y resuelva para encontrar los vectores propios asociados, las  $v$  de la ecuación 6.5.

Los valores propios de una matriz triangular, que incluye matrices diagonales, son sencillos de calcular porque el determinante de dicha matriz es simplemente el producto de la diagonal principal. Por ejemplo, para un  $4 \times 4$  matriz triangular, el determinante de la ecuación característica es

$$\det(A - I) = (a_{00} - )(a_{11} - )(a_{22} - )(a_{33} - )$$

que tiene cuatro raíces: los valores de la diagonal. Para triangular y diagonal matrices, las entradas en la diagonal principal son los valores propios.

Veamos un ejemplo de valor propio trabajado para la siguiente matriz:

$$U_n = \begin{bmatrix} 1 & 0 & 2 & -3 \end{bmatrix}$$

Seleccioné esta matriz para hacer las matemáticas más agradables, pero el proceso funciona para cualquier matriz. La ecuación característica significa que necesitamos los valores que hacen el determinante cero, como se muestra a continuación.

$$\begin{aligned}
 & 1 \\
 -2 -3 - & = (-)(-3 - ) + 2 \\
 & = 2 + 3 + 2 \\
 & = (+1)(+2)
 \end{aligned}$$

El polinomio característico se factoriza fácilmente para dar = -1, -2.

En código, para encontrar los valores propios y los vectores propios de una matriz, usamos np.linalg.eig. Revisemos nuestro cálculo anterior para ver si NumPy está de acuerdo:

---

```

>>> a = np.array([[0,1],[-2,-3]])
>>> print(np.linalg.eig(a)[0])
[-1. -2.]

```

---

La función np.linalg.eig devuelve una lista. El primer elemento es un vector de los valores propios de la matriz. El segundo elemento, que ignoramos por el momento, es una matriz cuyas columnas son los vectores propios asociados con cada uno de los valores propios. Tenga en cuenta que también podríamos haber usado np.linalg.eigvals para devolver solo los valores propios. De todos modos, vemos que nuestro cálculo de los valores propios de A es correcto.

Para encontrar los vectores propios asociados, volvemos a colocar cada uno de los valores propios en la ecuación 6.5 y resuelve para v. Por ejemplo, para = -1, obtenemos

$$(A - (-1)I)v = 0$$

$$\begin{matrix} 1 \\ [-2 -2] [v_0 v_1] = [0 0] \end{matrix}$$

lo que lleva al sistema de ecuaciones:

$$v_0 + v_1 = 0$$

$$-2v_0 - 2v_1 = 0$$

Este sistema tiene muchas soluciones, siempre que  $v_0 = -v_1$ . Eso significa que podemos elegir  $v_0$  y  $v_1$ , siempre que se preserve la relación entre ellos. Por lo tanto, tenemos nuestro vector propio para

$$= -1, v_0 = [1 -1]$$

Si repetimos este proceso para = -2, obtenemos la relación entre el componentes de  $v_2$  sean  $2v_0 = -v_1$ . Por lo tanto, seleccionamos  $v_1 = [-1 2]$  como el segundo vector propio.

Veamos si NumPy está de acuerdo con nosotros. Esta vez, mostraremos el segundo elemento de la lista devuelto por `np.linalg.eig`. Esta es una matriz donde las columnas de la matriz son los vectores propios:

---

```
>>> imprimir(np.linalg.eig(a)[1])
[[ 0.70710678 -0.4472136 ]
 [-0.70710678  0.89442719]]
```

---

Mmm . . . las columnas de esta matriz no parecen coincidir con nuestros vectores propios seleccionados. Pero no te preocupes, no cometimos ningún error. Recuerde que los vectores propios no se determinaron de forma única, sólo se determinó la relación entre los componentes. Si hubiéramos querido, podríamos haber seleccionado otros valores, siempre y cuando para un vector propio fueran de igual magnitud y signo opuesto, y para el otro estuvieran en una proporción de 2:1 con signos opuestos. Lo que devuelve NumPy es un conjunto de vectores propios de longitud unitaria. Entonces, para ver que nuestro cálculo manual es correcto, necesitamos hacer que nuestros vectores propios sean vectores unitarios dividiendo cada componente por la raíz cuadrada de la suma de los cuadrados de los componentes. El código es conciso, aunque un poco confuso:

---

```
>>> np.array([1,-1])/np.sqrt((np.array([1,-1])**2).sum())
array([ 0.70710678, -0.70710678])
>>> np.array([-1,2])/np.sqrt((np.array([-1,2])**2).sum())
array([-0.4472136,  0.89442719])
```

---

Ahora vemos que estamos en lo cierto. Las versiones de vectores unitarios de los vectores propios coinciden con las columnas de la matriz devuelta por NumPy.

Usaremos los vectores propios y los valores propios de una matriz a menudo cuando estemos haciendo aprendizaje profundo. Por ejemplo, los veremos nuevamente más adelante en el capítulo cuando investiguemos el análisis de componentes principales. Pero antes de que podamos aprender sobre PCA, necesitamos cambiar de enfoque una vez más y aprender sobre las normas vectoriales y las métricas de distancia comúnmente utilizadas en el aprendizaje profundo, especialmente sobre la matriz de covarianza.

## Normas vectoriales y métricas de distancia

En el lenguaje común del aprendizaje profundo, las personas son algo descuidadas y usan los términos norma y distancia indistintamente. Podemos perdonarlos por hacerlo; la diferencia entre los términos es pequeña en la práctica, como veremos a continuación.

Una norma vectorial es una función que asigna un vector, real o complejo, a algún valor,  $x \in \mathbb{C}^n$ ,  $x \geq 0$ . Una norma debe satisfacer algunas propiedades específicas en un sentido matemático, pero en la práctica, no todo lo que se llama norma es, de hecho, una norma. En el aprendizaje profundo, normalmente utilizamos normas como distancias entre pares de vectores. En la práctica, una propiedad importante de una medida de distancia es que el orden de las entradas no importa. Si  $f(x,y)$  es una distancia, entonces  $f(x,y) = f(y,x)$ . Una vez más, esto no se sigue rigurosamente; por ejemplo, a menudo verá la divergencia Kullback-Leibler (divergencia KL) utilizada como distancia aunque esta propiedad no se cumpla.

Comencemos con las normas vectoriales y veamos cómo podemos usarlas fácilmente como medida de distancia entre vectores. Luego introduciremos el importante concepto de matriz de covarianza, muy utilizada por sí sola en el aprendizaje profundo, y veremos cómo podemos crear una medida de distancia a partir de ella: la distancia de Mahalanobis. Terminaremos la sección presentando la divergencia KL, que podemos ver como una medida entre dos distribuciones de probabilidad discretas.

### Normas L y métricas de distancia

Para un vector de  $n$  dimensiones,  $x$ , definimos la norma  $p$  del vector como

$$\|x\|_p \equiv \left( \sum_i |x_i|^p \right)^{\frac{1}{p}} \quad (6.6)$$

donde  $p$  es un número real. Aunque utilizamos  $p$  en la definición, la gente generalmente se refiere a éstas como normas  $L_p$ . Vimos una de estas normas en el Capítulo 5 cuando definimos la magnitud de un vector. En ese caso, estábamos calculando la norma  $L_2$ ,

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{x^T x}$$

que es la raíz cuadrada del producto interno de  $x$  consigo mismo.

Las normas que utilizamos con más frecuencia en el aprendizaje profundo son la norma  $L_2$  y la norma  $L_1$ ,

$$\|x\|_1 = \sum_i |x_i|$$

que no es más que la suma de los valores absolutos de los componentes de  $x$ . Otra norma que encontrarás es la norma  $L^\infty$ ,

$$\|x\|_\infty = \max_i |x_i|$$

el valor absoluto máximo de los componentes de  $x$ .

Si reemplazamos  $x$  con la diferencia de dos vectores,  $x - y$ , podemos tratar las normas como medidas de distancia entre los dos vectores. Alternativamente, podemos imaginar el proceso calculando la norma vectorial en el vector que es la diferencia entre  $x$  e  $y$ .

Pasar de norma a distancia supone un cambio trivial en la ecuación 6.6:

$$\|x - y\|_p = \left( \sum_i |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (6.7)$$

La distancia  $L_2$  se convierte en

$$\|x - y\|_2 = \sqrt{(x_0 - y_0)^2 + (x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_{n-1} - y_{n-1})^2}$$

Esta es la distancia euclídea entre dos vectores. La distancia L1 a menudo se denomina distancia de Manhattan (también llamada distancia de cuadra de la ciudad, distancia de furgón o distancia de taxi):

$$L1 = \sum_i |x_i - y_i|$$

Se llama así porque corresponde a la longitud que recorrería un taxi en la cuadrícula de calles de Manhattan. La distancia  $L^\infty$  a veces se conoce como distancia de Chebyshev.

Las ecuaciones normativas tienen otros usos en el aprendizaje profundo. Por ejemplo, la caída de peso, utilizada en el aprendizaje profundo como regularizador, utiliza la norma L2 de los pesos del modelo para evitar que los pesos crezcan demasiado. La norma L1 de los pesos también se utiliza a veces como regularizador.

Pasemos ahora a considerar el importante concepto de matriz de covarianza. No es una métrica de distancia en sí misma, pero la usa una persona y aparecerá nuevamente más adelante en el capítulo.

### Matrices de covarianza

Si tenemos una colección de mediciones de múltiples variables, como un conjunto de entrenamiento con vectores de características, podemos calcular la varianza de las características entre sí. Por ejemplo, aquí hay una matriz de observaciones de cuatro variables, una por fila:

$$X = \begin{matrix} 5,1 & 3,5 & 1,4 & 0,2 \\ 4,9 & 3,0 & 1,4 & 0,2 \\ 4,7 & 3,2 & 1,3 & \\ 4,6 & 3,1 & 1,5 & 0,2 & 0,2 \\ 5,0 & 3,6 & 1,4 & 0,2 \end{matrix}$$

En realidad, X son las primeras cinco muestras del famoso conjunto de datos del iris. Para el conjunto de datos de iris, las características son mediciones de las partes de las flores de iris de tres especies diferentes. Puedes cargar este conjunto de datos en NumPy usando sklearn:

---

```
>>> desde sklearn importar conjuntos
de datos >>> iris = datasets.load_iris()
>>> X = iris.datos[:5]
>>> X
matriz([[5.1, 3.5, 1.4, 0.2], [4.9,
    3.0, 1.4, 0.2], [4.7, 3.2,
    1.3, 0.2], [4.6, 3.1, 1.5,
    0.2], [5.0, 3.6, 1.4, 0.2]])
```

---

Podríamos calcular la desviación estándar de cada una de las características, las columnas de A, pero eso sólo nos informaría sobre la varianza de los valores de esa característica alrededor de su media. Dado que tenemos múltiples funciones, sería bueno saber algo sobre cómo funcionan las funciones en, digamos, la columna cero y

La columna uno varía en conjunto. Para determinar esto, necesitamos calcular la matriz de covarianza. Esta matriz captura la variación de las características individuales a lo largo de la diagonal principal. Mientras tanto, los valores fuera de la diagonal representan cómo varía una característica a medida que varía otra: estas son las covarianzas. Como hay cuatro características, la matriz de covarianza, que siempre es cuadrada, es, en este caso, una matriz de  $4 \times 4$ . Encontramos los elementos de la matriz de covarianza, calculando

$$\Sigma_{ij} = \frac{1}{n - 1} \sum_{k=0}^{n-1} (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j) \quad (6.8)$$

suponiendo que las filas de la matriz, X, son las observaciones y las columnas de X representan las diferentes características. Las medias de las características en todas las filas son  $\bar{x}_i$  y  $\bar{x}_j$  para las características i y j. Aquí, n es el número de observaciones, el número de filas en X. Podemos ver que cuando  $i = j$ , el valor de la covarianza es la varianza normal para esa característica. Cuando  $i \neq j$ , el valor es cómo i y j varían juntos. A menudo denotamos la matriz de covarianza como y siempre es simétrica:  $\Sigma_{ij} = \Sigma_{ji}$ .

Calculemos algunos elementos de la matriz de covarianza para X anterior. El las medias por característica son  $\bar{x} = [4,86, 3,28, 1,4, 0,2]$ . Busquemos la primera fila de . Esto nos dirá la variación de la primera característica (columna de) y cómo esa característica varía con la segunda, tercera y cuarta característica. Por lo tanto, necesitamos calcular  $\Sigma_{00}$ ,  $\Sigma_{01}$ ,  $\Sigma_{02}$  y  $\Sigma_{03}$ :

$$\Sigma_{00} = \frac{1}{5 - 1} \sum_{k=0}^4 (x_{k0} - 4,86)(x_{k0} - 4,86) = 0,0430$$

$$\Sigma_{01} = \frac{1}{5 - 1} \sum_{k=0}^4 (x_{k0} - 4,86)(x_{k1} - 3,28) = 0,0365$$

$$\Sigma_{02} = \frac{1}{5 - 1} \sum_{k=0}^4 (x_{k0} - 4,86)(x_{k2} - 1,40) = -0,0025$$

$$\Sigma_{03} = \frac{1}{5 - 1} \sum_{k=0}^4 (x_{k0} - 4,86)(x_{k3} - 0,20) = 0,0$$

Podemos repetir este cálculo para todas las filas de para obtener la matriz de covarianza completa:

$$\begin{aligned} & 0,0430 \ 0,0365 \ -0,0025 \ 0,0000 \ 0,0365 \\ & = \ 0,0670 \ -0,0025 \ 0,0000 \\ & \quad -0,0025 \ -0,0025 \ 0,0050 \ 0,0000 \\ & \quad 0,0000 \ 0,0000 \ 0,0000 \ 0,0000 \end{aligned}$$

Los elementos a lo largo de la diagonal representan la variación de las características de X. Observe que para la cuarta característica de X, todas las varianzas y covarianzas son cero. Esto tiene sentido porque todos los valores de esta característica en X son lo mismo; no hay variación.

Podemos calcular la matriz de covarianza para un conjunto de observaciones en código. usando np.cov:

---

```
>>> imprimir(np.cov(X, filavar=False))
[[ 0,043 0,0365 -0,0025 0. ]
 [ 0,0365 0,067 -0,0025 0. ]
 [-0,0025 -0,0025 0,005 0. ]
 [ 0. ]]
```

---

Observe que la llamada a np.cov incluye rowvar=False. Por defecto, np.cov espera que cada fila de su argumento sea una variable y que las columnas sean las observaciones de esa variable. Esto es lo opuesto a la forma habitual en que un conjunto de observaciones generalmente se almacenan en una matriz para el aprendizaje profundo. por lo tanto, nosotros use la palabra clave rowvar para decirle a NumPy que las filas, no las columnas, son las observaciones.

Afirmé anteriormente que la diagonal de la matriz de covarianza devuelve el variaciones de las características en X. NumPy tiene una función, np.std, para calcular las desviación estándar, y elevar al cuadrado la salida de esta función debería darnos las variaciones de las características por sí mismas. Para X, obtenemos

---

```
>>> imprimir(np.std(X, eje=0)**2)
[0,0344 0,0536 0,004 0. ]
```

---

Estas varianzas no se parecen a la diagonal de la matriz de covarianza. La diferencia se debe al  $n - 1$  en el denominador de la covarianza. ecuación, Ecuación 6.8. De forma predeterminada, np.std calcula lo que se conoce como estimación sesgada de la varianza muestral. Esto significa que en lugar de dividir por  $n - 1$ , se divide por n. Para conseguir que np.std calcule el estimador insesgado de la varianza, necesitamos agregar la palabra clave ddof=1 ,

---

```
>>> imprimir(np.std(X, eje=0, ddof=1)**2)
[0,043 0,067 0,005 0. ]
```

---

entonces obtendremos los mismos valores que a lo largo de la diagonal de . Ahora que sabemos cómo calcular la matriz de covarianza, usémosla en un métrica de distancia.

### Distancia de Mahalanobis

Arriba, representamos un conjunto de datos mediante una matriz donde las filas del conjunto de datos son observaciones y las columnas son los valores de las variables que componen cada observación. En términos de aprendizaje automático, las filas son los vectores características. Como vimos arriba, podemos calcular la media de cada característica en todos las observaciones, y podemos calcular la matriz de covarianza. Con estos valores, podemos definir una métrica de distancia llamada distancia de Mahalanobis,

$$DM = \sqrt{(x - \bar{x})^T \Sigma^{-1} (x - \bar{x})} \quad (6.9)$$

donde  $x$  es un vector, es el vector formado por los valores medios de cada característica y es la matriz de covarianza. Tenga en cuenta que esta métrica utiliza la inversa de la matriz de covarianza, no de la matriz de covarianza en sí.

La ecuación 6.9 mide, en cierto sentido, la distancia entre un vector y una distribución con el vector medio. La dispersión de la distribución se captura en  $\Sigma$ . Si no hay covarianza entre las características en la conjunto de datos y cada característica tiene la misma desviación estándar, luego se convierte en la matriz identidad, que es su propia inversa. En ese caso, sale de la  $\Sigma^{-1}$  efectivamente Ecuación 6.9 y la distancia de Mahalanobis se convierte en L2-distancia (distancia euclídea).

Otra forma de pensar en la distancia de Mahalanobis es reemplazarla con otro vector, llámelo  $y$ , que proviene del mismo conjunto de datos que  $x$ . Entonces  $DM$  es la distancia entre los dos vectores, tomando la varianza del conjunto de datos en cuenta.

Podemos usar la distancia de Mahalanobis para construir un clasificador simple. Si, dado un conjunto de datos, calculamos el vector de características medio de cada clase en el conjunto de datos (este vector también se llama centroide), podemos usar el Mahalanobis distancia para asignar una etiqueta a un vector de características desconocido,  $x$ . Podemos hacerlo por calcular todas las distancias de Mahalanobis a los centroides de clase y asignar  $x$  a la clase que devuelve el valor más pequeño. Este tipo de clasificador a veces se denomina clasificador de centroide más cercano y, a menudo, lo verá implementado. utilizando la distancia L2 en lugar de la distancia de Mahalanobis. Podría decirse que tú podemos esperar que la distancia de Mahalanobis sea la mejor métrica porque toma en cuenta la variación del conjunto de datos.

Usemos el conjunto de datos sobre cáncer de mama incluido con sklearn para construir el Clasificador de centroide más cercano utilizando la distancia de Mahalanobis. El conjunto de datos sobre cáncer de mama tiene dos clases: benigno (0) y maligno (1). El conjunto de datos contiene 569 observaciones, cada una de las cuales tiene 30 características derivadas de la histología. diapositivas. Construiremos dos versiones del clasificador de centroide más cercano: una que use la distancia de Mahalanobis y el otro usando la distancia euclídea. Nuestro La expectativa es que el clasificador que utiliza la distancia de Mahalanobis realice mejor.

El código que necesitamos es sencillo:

---

```
importar numpy como np
desde sklearn importar conjuntos de datos
de scipy.spatial.distance importar mahalanobis
```

```
bc = conjuntos de datos.load_breast_cancer()
d = bc.datos; l = bc.objetivo
i = np.argsort(np.random.random(len(d)))
d = d[i]; l = l[i]
xtrn, ytrn = d[:400], l[:400]
xtst, ytst = d[400:], l[400:]
```

```

i = np.donde(ytrn == 0)
m0 = xtrn[i].media(eje=0) i
= np.donde(ytrn == 1) m1
= xtrn[i].media(eje=0)
S = np.cov(xtrn, filavar=False)
SI= np.linalg.inv(S)

puntuación def(xtst, ytst, m, SI):
    nc = 0
    para i en rango(len(ytst)): d
        = np.array([mahalanobis(xtst[i],m[0],SI),
                    mahalanobis(xtst[i],m[1],SI)])
    c = np.argmin(d)
    si (c == ytst[i]):
        nc += 1
    devolver nc/len(ytst)

mscore = puntaje(xtst, ytst, [m0,m1], SI)
escore = puntaje(xtst, ytst, [m0,m1], np.identity(30))
print("Puntuación de Mahalanobis = %0.4f" %
      mscore ) print("Puntuación euclidiana = %0.4f" % puntuación)

```

---

Comenzamos importando los módulos que necesitamos, incluidos mahalanobis de Ciencia ficción . Esta función acepta dos vectores y la inversa de una matriz de covarianza y devuelve DM. Obtenemos el siguiente conjunto de datos en d con etiquetas en I. Aleatorizamos el orden y extraemos las primeras 400 observaciones como datos de entrenamiento (xtrn) con etiquetas (ytrn). Retenemos las observaciones restantes para realizar pruebas (xtst, ytst).

Entrenamos el modelo a continuación. El entrenamiento consiste en extraer todas las observaciones pertenecientes a cada clase y calcular m0 y m1. Estos son los valores medios de cada una de las 30 características para todas las observaciones de clase 0 y clase 1.

Luego calculamos la matriz de covarianza de todo el conjunto de entrenamiento (S) y su inversa (SI).

La función de puntuación toma las observaciones de la prueba, una lista de los vectores medios de clase y la inversa de la matriz de covarianza. Revisa cada observación de prueba y calcula las distancias de Mahalanobis (d). Luego utiliza la distancia más pequeña para asignar la etiqueta de clase (c). Si la etiqueta asignada coincide con la etiqueta de prueba real, la contamos (nc). Al final de la función, devolvemos la precisión general.

Llamamos a la función de puntuación dos veces. La primera llamada utiliza la matriz de covarianza inversa (SI), mientras que la segunda llamada utiliza una matriz de identidad, lo que hace que la puntuación calcule la distancia euclidiana. Finalmente, imprimimos ambos resultados.

La aleatorización del conjunto de datos significa que cada vez que se ejecuta el código, generará puntuaciones ligeramente diferentes. Al ejecutar el código 100 veces se obtienen las siguientes puntuaciones medias ( $\pm$  la desviación estándar).

Distancia	Puntuación media
Mahalanobis $0,9595 \pm 0,0142$	
Euclíadiano $0,8914 \pm 0,0185$	

Esto muestra claramente que el uso de la distancia de Mahalanobis conduce a un mejor rendimiento del modelo, con una mejora de aproximadamente el 7 por ciento en la precisión.

Un uso reciente de la distancia de Mahalanobis en el aprendizaje profundo es tomar los valores de la capa de incrustación de nivel superior, un vector y use la distancia de Mahalanobis para detectar entradas fuera del dominio o adversarias. Una entrada fuera del dominio es aquella que es significativamente diferente del tipo de datos para el cual el modelo fue entrenado. Una entrada adversaria es aquella en la que un adversario intenta deliberadamente engañar al modelo proporcionando una entrada que no es de clase X pero que el modelo etiquetará como clase X.

#### Divergencia de Kullback-Leibler La

divergencia de Kullback-Leibler (divergencia KL), o entropía relativa, es una medida de la similitud entre dos distribuciones de probabilidad: cuanto menor es el valor, más similares son las distribuciones.

Si P y Q son distribuciones de probabilidad discretas, la divergencia KL es

$$DKL(P \parallel Q) = \sum x P(x) \log_2 \left( \frac{P(x)}{Q(x)} \right)$$

donde log2 es el logaritmo en base 2. Ésta es una medida teórica de la información; la salida está en bits de información. A veces se utiliza el logaritmo natural, ln, en cuyo caso se dice que la medida está en nats. La función SciPy que implementa la divergencia KL está en scipy.special como rel\_entr. Tenga en cuenta que rel\_entr utiliza el registro natural, no el registro base-2. Tenga en cuenta también que la divergencia KL no es una métrica de distancia en el sentido matemático porque viola la propiedad de simetría,  $DKL(P \parallel Q) \neq DKL(Q \parallel P)$ , pero eso no impide que la gente la use como tal. de vez en cuando.

Veamos un ejemplo de cómo podríamos usar la divergencia KL para medir entre diferentes distribuciones de probabilidad discretas. Mediremos la divergencia entre dos distribuciones binomiales diferentes y una distribución uniforme. Luego, trazaremos las distribuciones para ver si, visualmente, creemos en los números.

Para generar las distribuciones, tomaremos muchos sorteos de una distribución uniforme con 12 salidas posibles. Podemos hacer esto rápidamente en código usando np.random.randint. Luego, haremos extracciones de dos distribuciones binomiales diferentes,  $B(12, 0,4)$  y  $B(12, 0,9)$ , lo que significa 12 ensayos con probabilidades de 0,4 y 0,9 por ensayo. Generaremos histogramas de los sorteos resultantes, los dividiremos por la suma de los recuentos y usaremos los histogramas reescalados como nuestras distribuciones de probabilidad. Entonces podemos medir las divergencias entre ellos.

El código que necesitamos es

---

```
de scipy.special importar rel_entr
= 1000000
p = np.random.randint(0,13,tamaño=N)
p = np.bincount(p) p =
p / p.sum() q =
np.random.binomial(12,0.9,tamaño= N) q =
np.bincount(q) q = q /
q.sum() w =
np.random.binomial(12,0.4,size=N) w =
np.bincount(w) w =
w / w.sum ()
imprimir(rel_entr(q,p).sum())
imprimir(rel_entr(w,p).sum())
```

---

Cargamos `rel_entr` desde SciPy y establecemos el número de sorteos para cada distribución en 1.000.000 (N). El código para generar las respectivas distribuciones de probabilidad sigue el mismo método para cada distribución. Extraemos N muestras de la distribución, comenzando con el uniforme . Usamos `randint` porque devuelve números enteros en el rango [0, 12] para que podamos hacer coincidir los valores discretos [0, 12] que devuelve binomial durante 12 intentos. Obtenemos el histograma del vector de sorteos usando `np.bincount`. Esta función cuenta la frecuencia de valores únicos en un vector . Finalmente, cambiamos los conteos a fracciones dividiendo el histograma por la suma . Esto nos da un vector de 12 elementos en `p` que representa la probabilidad de que `randint` devuelva los valores del 0 al 12. Suponiendo que `randint` utiliza un buen generador de números pseudoaleatorios, esperamos que las probabilidades sean aproximadamente iguales para cada valor en `p`. (NumPy utiliza el generador de números pseudoaleatorios Mersenne Twister, uno de los mejores que existen, por lo que estamos seguros de que obtendremos buenos resultados).

Repetimos este proceso, sustituyendo `binomial` por `randint`, tomando muestras de distribuciones binomiales usando probabilidades de 0,9 y 0,4. Nuevamente, histogramar los sorteos y convertir los conteos a fracciones nos da las distribuciones de probabilidad restantes, `q` y `w`, basadas en 0,9 y 0,4, respectivamente.

Finalmente estamos listos para medir la divergencia. La función `rel_entr` es un poco diferente de otras funciones en que no devuelve DKL directamente. En cambio, devuelve un vector de la misma longitud que sus argumentos, donde cada elemento del vector es parte de la suma general que conduce a DKL. Por lo tanto, para obtener el número de divergencia real, necesitamos sumar los elementos de este vector. Entonces, imprimimos la suma de la salida de `rel_entr`, comparando las dos distribuciones binomiales con la distribución uniforme.

La naturaleza aleatoria de los sorteos significa que obtendremos números ligeramente diferentes cada vez que ejecutemos el código. Una carrera dio

---

Divergencia de distribuciones	
DKL(Q P)	1,1826
DKL(W P)	0,6218

---

Esto muestra que la distribución binomial con probabilidad de 0,9 diverge más de una distribución uniforme que la distribución binomial con probabilidad de 0,4. Recuerde que cuanto menor es la divergencia, más cercanas están entre sí las dos distribuciones de probabilidad.

¿Creemos en este resultado? Una forma de comprobarlo es visualmente, trazando las tres distribuciones y viendo si  $B(12, 0,4)$  se parece más a una distribución uniforme que  $B(12, 0,9)$ . Esto lleva a la Figura 6-1.

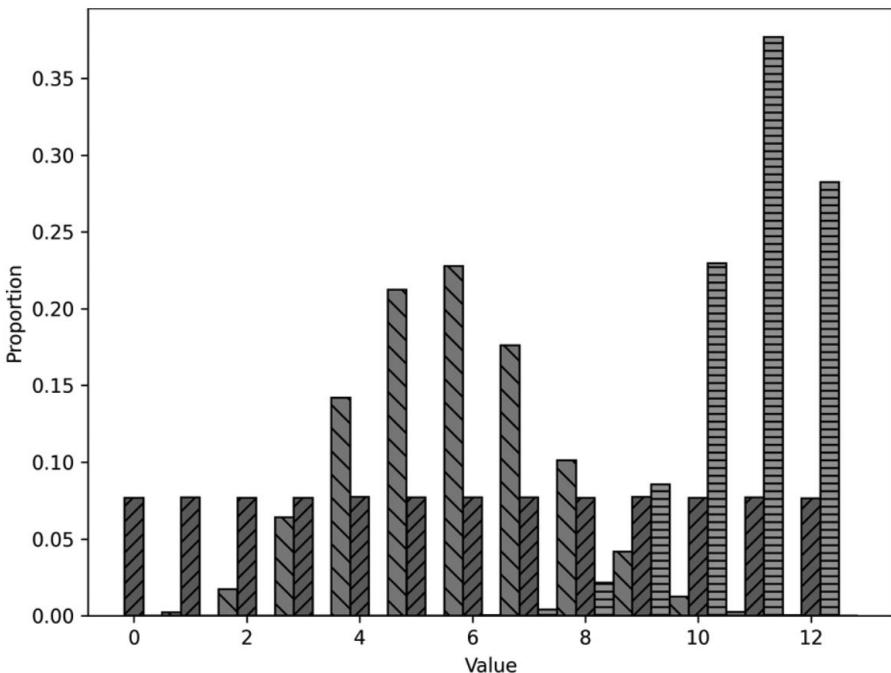


Figura 6-1: Tres distribuciones de probabilidad discretas diferentes: uniforme (hash directo),  $B(12,0.4)$  (hash hacia atrás) y  $B(12,0.9)$  (hash horizontal)

Aunque está claro que ninguna distribución binomial es particularmente uniforme, la distribución  $B(12, 0,4)$  está relativamente centrada en el rango y se distribuye en más valores que la distribución  $B(12, 0,9)$ . Parece razonable pensar que  $B(12, 0,4)$  se parece más a la distribución uniforme, que es precisamente lo que nos indicó la divergencia KL al devolver un valor menor.

Ahora tenemos todo lo que necesitamos para implementar el análisis de componentes principales.

## Análisis de componentes principales

Supongamos que tenemos una matriz,  $X$ , que representa un conjunto de datos. Entendemos que la variación de cada una de las características no tiene por qué ser la misma. Si pensamos en cada observación como un punto en un espacio  $n$ -dimensional, donde  $n$  es el número de características en cada observación, podemos imaginar una nube de puntos con una cantidad diferente de dispersión en diferentes direcciones.

El análisis de componentes principales (PCA) es una técnica para aprender las direcciones de la dispersión en el conjunto de datos, comenzando con la dirección alineada a lo largo de la mayor dispersión. Esta dirección se llama componente principal. Luego encontrará los componentes restantes en orden de dispersión decreciente, con cada nuevo componente ortogonal a todos los demás. La parte superior de la Figura 6-2 muestra un conjunto de datos 2D y dos flechas. Sin saber nada sobre el conjunto de datos, podemos ver que la flecha más grande apunta en la dirección de mayor dispersión. Esto es lo que queremos decir con el componente principal.

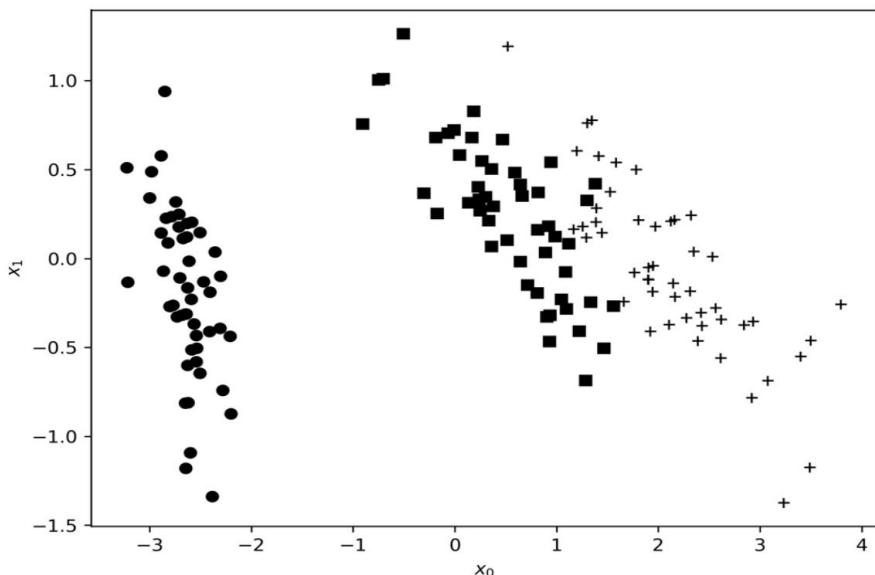
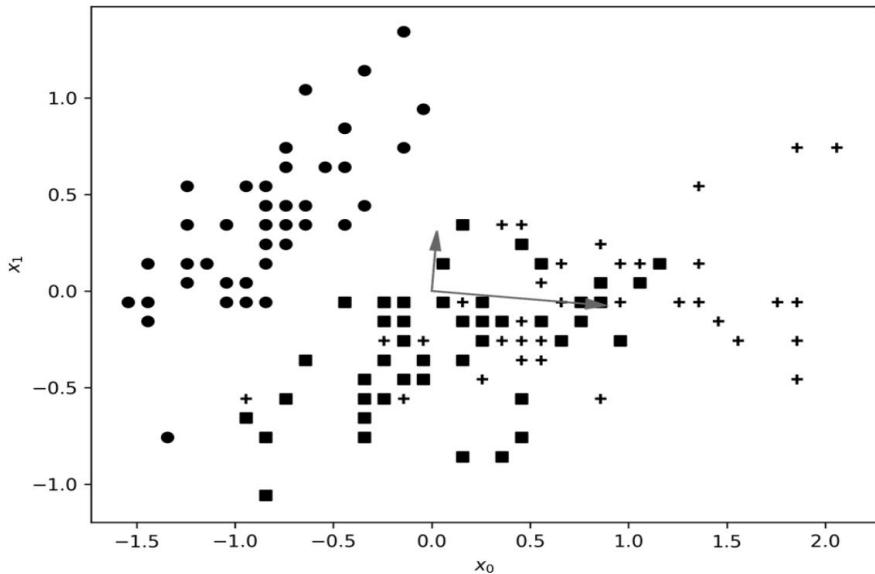


Figura 6-2: Las dos primeras características del conjunto de datos del iris y las direcciones de los componentes principales (arriba), y el conjunto de datos del iris después de la transformación mediante PCA (abajo)

A menudo utilizamos PCA para reducir la dimensionalidad de un conjunto de datos.

Si hay 100 variables por observación, pero los dos primeros componentes principales explican el 95 por ciento de la dispersión de los datos, entonces mapear el conjunto de datos a lo largo de esos dos componentes y descartar los 98 componentes restantes podría caracterizar adecuadamente el conjunto de datos con solo dos variables. También podemos usar PCA para aumentar un conjunto de datos, asumiendo características continuas.

Entonces, ¿cómo funciona la PCA? Todo este discurso sobre la dispersión de los datos implica que la PCA podría utilizar la matriz de covarianza  $y$ , de hecho, lo hace. Podemos dividir el algoritmo PCA en unos pocos pasos:

1. Significa centrar los datos.
2. Calcule la matriz de covarianza, , de los datos centrados en la media.
3. Calcule los valores propios y los vectores propios de .
4. Ordene los valores propios disminuyendo el valor absoluto.
5. Descartar los valores propios/vectores propios más débiles (opcional).
6. Forme una matriz de transformación,  $W$ , utilizando los vectores propios restantes.
7. Genere nuevos valores transformados a partir del conjunto de datos existente,  $x' = Wx$ . A veces se las denomina variables derivadas.

Analicemos un ejemplo de este proceso utilizando el conjunto de datos de iris (Listado 6-1). Reduciremos la dimensionalidad de los datos de cuatro características a dos. Primero el código, luego la explicación:

---

```
de sklearn.datasets importar load_iris
iris = load_iris().data.copy()
m = iris.mean(axis=0)
ir = iris - m

cv = np.cov(ir, rowvar=False)
val, vec = np.linalg.eig(cv)
val = np.abs(val)
idx = np.argsort(val)[-1]
ex = val[idx] / val.sum()
print("fracción explicada: ", ej)
w = np.vstack((vec[:,idx[0]],vec[:,idx[1]]))

d = np.zeros((ir.shape[0],2)) para i
en el rango(ir.shape[0]): d[i,:] =
    np.dot(w,ir[i])
```

---

Listado 6-1: Análisis de componentes principales (PCA)

Comenzamos cargando el conjunto de datos del iris, cortesía de sklearn. Esto nos da el iris como una matriz de  $150 \times 4$ , ya que hay 150 observaciones, cada una con cuatro características. Calculamos el valor medio de cada característica y lo restamos del conjunto de datos, confiando en las reglas de transmisión de NumPy para restar  $m$  de cada fila de iris. Trabajaremos con la matriz centrada en la media en el futuro.

El siguiente paso es calcular la matriz de covarianza . La salida,  $cv$ , es una matriz de  $4 \times 4$ , ya que tenemos cuatro características por observación. Seguimos esto

calculando los valores propios y los vectores propios de `cv` y luego tomando el valor absoluto de los valores propios para obtener la magnitud. Queremos que los valores propios estén en orden de magnitud decreciente, por lo que obtenemos los índices que los ordenan de esa manera usando el modismo de Python de `[::-1]` para invertir el orden de una lista o matriz.

La magnitud de los valores propios es proporcional a la fracción de los variaciones en el conjunto de datos a lo largo de cada componente principal; por lo tanto, si escalamos los valores propios por su suma global, obtenemos la proporción explicada por cada componente principal (ex). La fracción de varianza explicada es

---

fracción explicada: [0,92461872 0,05306648 0,01710261 0,00521218]

---

lo que indica que dos componentes principales explican casi el 98 por ciento de la variación en el conjunto de datos del iris. Por lo tanto, sólo mantendremos los dos primeros componentes principales en el futuro.

Creamos la matriz de transformación, `w`, a partir de los vectores propios que van con los dos valores propios más grandes. Recuerde, `eig` devuelve los vectores propios como las columnas de la matriz `vec`. La matriz de transformación, `w`, es una matriz de  $2 \times 4$  porque asigna un vector de características de cuatro componentes a un nuevo vector de dos componentes.

Todo lo que queda es crear un lugar para guardar las observaciones transformadas y completarlas. El nuevo conjunto de datos de dimensión reducida está en `d`. Ahora podemos trazar todo el conjunto de datos transformado, etiquetando cada punto según la clase a la que pertenece. El resultado es la parte inferior de la Figura 6-2.

En la parte superior de la Figura 6-2 hay un gráfico del conjunto de datos original utilizando solo las dos primeras características. Las flechas indican los dos primeros componentes principales, y el tamaño de las flechas muestra qué parte de la varianza en los datos explican estos componentes. El primer componente explica la mayor parte de la variación, lo que tiene sentido visualmente.

En este ejemplo, las variables derivadas en la parte inferior de la Figura 6-2 han hecho que sea más fácil trabajar con el conjunto de datos, ya que las clases están mejor separadas que en la parte superior usando solo dos de las características originales. A veces, PCA facilita el aprendizaje de un modelo debido al tamaño reducido del vector de características. Sin embargo, este no es siempre el caso. Durante la PCA, es posible que pierda una característica fundamental que permite la separación de clases. Como ocurre con la mayoría de las cosas en el aprendizaje automático, la experimentación es vital.

PCA se utiliza comúnmente y, por lo tanto, cuenta con soporte en múltiples kits de herramientas. En lugar de la docena de líneas de código que usamos anteriormente, podemos lograr lo mismo usando la clase PCA del módulo `sklearn.decomposition`:

---

```
de sklearn.decomposition importar PCA
pca = PCA(n_components=2)
pca.fit(ir) d
= pca.fit_transform(ir)
```

---

El nuevo conjunto de datos de dimensión reducida está en `d`. Al igual que otras clases de `sklearn`, después de decirle a PCA cuántos componentes queremos que aprenda, utiliza `fit` para configurar la matriz de transformación (`w` en el Listado 6-1). Luego aplicamos la transformación llamando a `fit_transform`.

## Descomposición de valores singulares y pseudoinversa

Terminaremos este capítulo con una introducción a la descomposición de valores singulares (SVD). Esta es una técnica poderosa para factorizar cualquier matriz en el producto de tres matrices, cada una con propiedades especiales. La derivación de SVD está más allá del alcance de este libro. Confío en que los lectores motivados profundicen en la vasta literatura sobre álgebra lineal para encontrar una presentación satisfactoria de dónde proviene la SVD y cómo se entiende mejor.

Nuestro objetivo es más modesto: familiarizarnos con las matemáticas que se encuentran en el aprendizaje profundo. Por lo tanto, nos contentaremos con la definición de SVD, una idea de lo que nos ofrece, algunos de sus usos y cómo trabajar con él en Python. Para el aprendizaje profundo, lo más probable es que encuentre SVD al calcular la pseudoinversa de una matriz no cuadrada. También veremos cómo funciona eso en esta sección.

La salida de SVD para una matriz de entrada, `A`, con elementos y forma reales,  $m \times n$ , donde  $m$  no necesariamente es igual a  $n$  (aunque podría) es

$$A = \text{ultravioleta} \quad (6.10)$$

`A` se ha descompuesto en tres matrices: `U`, `y` `V`. Tenga en cuenta que a veces puede ver que `V` es la escrita como  $V^*$ , la transpuesta conjugada de `V`. Esta forma más general que funciona con matrices de valores complejos. Nos limitaremos a matrices de valores reales, por lo que solo necesitamos la transposición de matrices ordinarias.

El SVD de una matriz  $m \times n$ , `A`, devuelve lo siguiente: `U`, que es  $m \times m$  y ortogonal; `, que es  $m \times n$  y diagonal; y V, que es  $n \times n$  y ortogonal. Recuerde que la transpuesta de una matriz ortogonal es su inversa, por lo que  $UU^* = I_m$  y  $VV^* = I_n$ , donde el subíndice de la matriz identidad da el orden de la matriz,  $m \times m$  o  $n \times n$ .`

En este punto del capítulo, es posible que haya levantado una ceja ante el , estado, que es  $m \times n$  y diagonal", ya que sólo hemos considerado que las matrices cuadradas son diagonales. Aquí, cuando decimos diagonal, nos referimos a una matriz diagonal rectangular. Esta es la extensión natural de una matriz diagonal, donde los elementos de lo que sería la diagonal son distintos de cero y todos los demás son cero. Por ejemplo,

$$m = \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \end{matrix}$$

es una matriz diagonal rectangular de  $3 \times 5$  porque sólo la diagonal principal es distinta de cero. El "singular" en la "descomposición de valores singulares" proviene del hecho de que los elementos de la matriz diagonal, son los valores singulares, las raíces cuadradas de los valores propios positivos de la matriz `A.TA`.

## SVD en acción

Seamos explícitos y usemos SVD para descomponer una matriz. Nuestra matriz de prueba es

$$A = [3 \ 2 \ 2 \ 3 \ -2]$$

Mostraremos la SVD en acción mediante una serie de pasos. Para obtener el SVD, usamos svd de scipy.linalg,

---

```
>>> from scipy.linalg import svd
a = np.array([[3,2,2],[2,3,-2]]) >>> u,s,vt
= svd(a)
```

---

donde u es U, vt es V , y s contiene los valores singulares:

---

```
>>>
print(u) [[-0.70710678
-0.70710678] [-0.70710678]
0.70710678]]
>>>
print(s) [5.
3.] >>> print(vt) [[-7.07106781e-01 -7.07106781e-01
-5.55111512e-17] [-2.35702260e-01 2.35702260e-01
-9.42809042e-01] [-6.66666667e-01 6.66666667e-01 3.33333333e-01]]
```

---

Comprobemos que los valores singulares son efectivamente las raíces cuadradas de los valores propios positivos de A T A:

---

```
>>> print(np.linalg.eig(aT @ a)[0])
[2.5000000e+01 5.0324328e-15 9.0000000e+00]
```

---

Esto nos muestra que sí, 5 y 3 son las raíces cuadradas de 25 y 9. Recuerde que eig devuelve una lista, cuyo primer elemento es un vector de valores propios.

Tenga en cuenta también que hay un tercer valor propio: cero. Quizás se pregunte: “¿Qué valor numérico debemos interpretar como cero?” Ésa es una buena pregunta que no tiene una respuesta definitiva y rápida. Normalmente, interpreto que los valores inferiores a 10-9 son cero.

La afirmación de SVD es que U y V son matrices unitarias. Si es así, sus productos Los productos con sus transpuestas deberían ser la matriz identidad:

---

```
>>> print(uT @
u) [[1.0000000e+00 3.33066907e-16]
[3.33066907e-16 1.0000000e+00]]
>>> print(vt @
v.T) [[ 1.0000000e+00 8.00919909e-17
-1.85037171e-17] [ 8.00919909e-17 1.0000000e+00
-5.55111512e-17] [-1.85037171e-17 -5.55111512e-17 1.0000000e+00]]
```

---

Dado el comentario anterior sobre los valores numéricos que deberíamos interpretar como cero, esta es de hecho la matriz identidad. Observe que svd devolvió V. Sin embargo, , no V. dado que (V ) = V, todavía estamos multiplicando V .

La función svd no devuelve más que los valores diagonales de . Por lo tanto, reconstruyamos y usémoslo para ver que SVD funciona, lo que significa que podemos usar U, , y V para recuperar A:

---

```
>>> S = np.ceros((2,3))
>>> S[0,0], S[1,1] = s >>>
imprimir(S) [[5.
0. 0.] [0. 3.
0.]]
>>> A = u @ S @ vt
>>> imprimir(A)
[[ 3. 2. 2.] [ 2. 3.
-2.]]
```

---

Esta es la A con la que empezamos, casi: la A recuperada ya no es de tipo entero, un cambio sutil que vale la pena recordar al escribir código.

## Two Applications

SVD es un lindo truco, pero ¿qué podemos hacer con él? La respuesta corta es "mucho".

Veamos dos aplicaciones. El primero es utilizar SVD para PCA. La clase PCA de sklearn que utilizamos en la sección anterior utiliza SVD internamente. El segundo ejemplo aparece en el aprendizaje profundo: usar SVD para calcular la pseudoinversa de Moore-Penrose, una generalización de la inversa de una matriz cuadrada a matrices  $m \times n$ .

### SVD para PCA

Para ver cómo usar SVD para PCA, usemos los datos del iris de la sección anterior para poder comparar con esos resultados. La clave es truncar las matrices y para mantener solo el número  $V$  deseado de valores singulares más grandes. El

El código de descomposición colocará los valores singulares en orden decreciente a lo largo de la diagonal de. Para nosotros, solo necesitamos conservar las primeras  $k$  columnas de.<sup>En</sup> código, entonces,

---

```
u,s,vt = svd(ir)    S
= np.zeros((ir.shape[0], ir.shape[1])) para i en el
rango(4): S[i,i] = s[ i]
S = S[:, :2]
```

---

$T = tu @ S$

---

Aquí estamos usando ir del Listado 6-1. Esta es la versión centrada en la media de la matriz del conjunto de datos del iris, con 150 filas de cuatro características cada una. Una llamada a svd nos da la descomposición de ir. Las siguientes tres líneas crean la matriz completa en S. Debido a que el conjunto de datos de iris tiene cuatro características, el vector s que devuelve svd tendrá cuatro valores singulares.

El truncamiento se produce manteniendo las dos primeras columnas de S . Al hacer esto se cambia de una matriz de  $150 \times 4$  a una matriz de  $150 \times 2$ . Multiplicando U por

lo nuevo nos da el conjunto de datos del iris transformado. Dado que  $U$  es  $150 \times 150$  y  $V$  es  $150 \times 2$ , obtenemos un conjunto de datos de  $150 \times 2$  en  $T$ . Si trazamos esto como  $T[:,0]$  versus  $T[:,1]$ , obtenemos exactamente el mismo gráfico que la parte inferior de la Figura 6-2.

El pseudoinverso de Moore-Penrose

Como prometimos, nuestra segunda aplicación es calcular  $A^+$ , el Moore-Penrose pseudoinversa de una matriz  $A$  de  $m \times n$ . La matriz  $A^+$  se llama pseudoinverso porque, en conjunción con  $A$ , actúa como un inverso en ese

$$AA^+ = A \quad (6.11)$$

donde  $AA^+$  es algo así como la matriz identidad, lo que hace que  $A^+$  algo así como la inversa de  $A$ .

Sabiendo que la pseudoinversa de una matriz diagonal rectangular es simplemente el recíproco de los valores de la diagonal, dejando ceros como cero, seguidos de un transpuesta, podemos calcular la pseudoinversa de cualquier matriz general como

$$A^+ = V^T U^T$$

para  $A = UV^T$ , el SVD de  $A$ . Aviso, estamos usando la transpuesta conjugada,  $V^T$ , en lugar de la transpuesta ordinaria,  $V$ . Si  $A$  es real, entonces el ordinario La transposición es lo mismo que la transposición conjugada.

Veamos si la afirmación sobre  $A^+$  es verdad. Empezaremos con la matriz  $A$ . usamos en la sección anterior, calculamos el SVD y usamos las partes para encontrar la pseudoinversa. Finalmente, validaremos la ecuación 6.11.

Comenzaremos con  $A$ , la misma matriz que usamos arriba para el ejemplo de SVD:

---

```
>>> A = np.matriz([[3,2,2],[2,3,-2]])
>>> imprimir(A)
[[3 2 2]
 [2 3-2]]
```

---

Aplicando SVD nos dará  $U$  y  $V^T$  junto con la diagonal de  $\Sigma$ . usaremos los elementos diagonales a construir  $\Sigma^+$  manualmente.  $\Sigma^+$  es la transpuesta, Recuerde, donde los elementos diagonales que no son cero se cambian a sus recíprocos:

---

```
>>> u,s,vt = svd(A)
>>> Splus = np.array([[1/s[0],0],[0,1/s[1]],[0,0]])
>>> imprimir(Sobre)
[[0.2          0.          ]
 [0.          0.33333333]
 [0.          0.          ]]
```

---

Ahora podemos calcular  $A^+$  y verificar que  $AA^+ = A$ :

---

```
>>> Aplus = vt.T @ Splus @ uT
>>> imprimir(Aplus)
```

---

```

[[ 0.15555556 0.04444444]
 [ 0.04444444 0.15555556]
 [ 0.22222222 -0.22222222]]
>>> imprimir(A @ Aplus @
A) [[ 3. 2. 2.] [ 2.
3. -2.]]

```

---

Y, en este caso,  $AA^+$  es la matriz identidad:

---

```

>>> imprimir(A @
Aplus) [[1.00000000e+00
5.55111512e-17] [1.66533454e-16 1.00000000e+00]]

```

---

Esto concluye nuestra rápida mirada a la SVD y nuestra discusión sobre el álgebra lineal. Apenas hemos arañado la superficie, pero hemos cubierto lo que necesitamos saber.

## Resumen

Este capítulo pesado y el Capítulo 5 anterior abarcaron mucha álgebra lineal. Como tema matemático, el álgebra lineal es mucho más rico que nuestra presentación aquí.

Centramos el capítulo en las matrices cuadradas, ya que tienen un lugar especial en el álgebra lineal. Específicamente, analizamos las propiedades generales de las matrices cuadradas, con ejemplos. Aprendimos sobre los valores propios y los vectores propios, cómo encontrarlos y por qué son útiles. A continuación, analizamos las normas vectoriales y otras formas de medir la distancia, tal como aparecen a menudo en el aprendizaje profundo. Finalmente, terminamos el capítulo aprendiendo qué es PCA y cómo funciona, seguido de una mirada a la descomposición de valores singulares, con dos aplicaciones relevantes para el aprendizaje profundo.

El siguiente capítulo cambia de tema y cubre el cálculo diferencial. Esta es, afortunadamente, la parte “fácil” del cálculo y, en general, es todo lo que necesitamos para comprender los algoritmos específicos del aprendizaje profundo. Así que abróchense los cinturones de seguridad, asegúrese de que sus brazos y piernas estén completamente dentro del vehículo y prepárese para partir hacia el mundo del cálculo diferencial.



# 7

## CALCULO DIFERENCIAL



El descubrimiento del “cálculo” por Sir Issac Newton, y por separado por Gottfried Wilhelm Leibniz, fue uno de los mayores logros en la historia de las matemáticas.

El cálculo suele dividirse en dos partes principales: diferencial e integral. El cálculo diferencial habla de tasas de cambio y sus relaciones, plasmadas en la noción de derivada. El cálculo integral se ocupa de aspectos como el área bajo una curva.

No necesitamos cálculo integral para el aprendizaje profundo, pero usaremos cálculo diferencial con frecuencia. Por ejemplo, utilizamos cálculo diferencial para entrenar redes neuronales; Ajustamos los pesos de una red neuronal mediante el descenso de gradiente, que se basa en derivadas calculadas mediante el algoritmo de retropropagación.

El derivado será la estrella de este capítulo. Comenzaremos introduciendo la idea de pendiente y viendo cómo conduce a la noción de derivada. Luego definiremos formalmente la derivada y aprenderemos a calcular derivadas de funciones de una variable. Después de eso, aprenderemos a usar derivadas para encontrar los mínimos y máximos de funciones. Luego vienen las derivadas parciales, las derivadas con respecto a una sola variable para funciones de más de una variable. Usaremos ampliamente derivadas parciales en la retropropagación.

algoritmo. Concluiremos con los gradientes, que nos introducirán en el cálculo matricial, el tema del Capítulo 8.

## Pendiente

En la clase de álgebra aprendimos todo sobre las líneas. Una forma de definir una recta es la forma pendiente-intersección,

$$y = mx + b$$

donde  $m$  es la pendiente y  $b$  es la intersección  $y$ , el lugar donde la línea cruza el eje  $y$ . Nos interesa la pendiente. Si conocemos dos puntos de la recta,  $(x_1, y_1)$  y  $(x_0, y_0)$ , conocemos la pendiente de la recta:

$$\text{metro} = \frac{y_1 - y_0}{x_1 - x_0}$$

La pendiente nos dice cuánto cambio en  $y$  obtendremos para cualquier cambio dado en la posición  $x$ . Si la pendiente es positiva, entonces un cambio positivo en  $x$  conduce a un cambio positivo en  $y$ . Por otro lado, una pendiente negativa significa que un cambio positivo en  $x$  conduce a un cambio negativo en  $y$ .

La forma pendiente-intersección de la recta nos dice que la pendiente es la constante de proporcionalidad entre  $x$  y  $y$ . La intersección,  $b$ , es un desplazamiento constante. Esto significa que un cambio en la posición  $x$  de  $x_1$  a  $x_0$  conduce a un cambio  $m(x_1 - x_0)$  en  $y$ .

Las pendientes relacionan dos cosas y nos dicen cómo el cambio de una afecta a la otra. Volveremos a esta idea varias veces en el libro.

Ahora, visualicemos esto con algunos ejemplos. La figura 7-1 muestra la gráfica de una curva y unas rectas que la cortan.

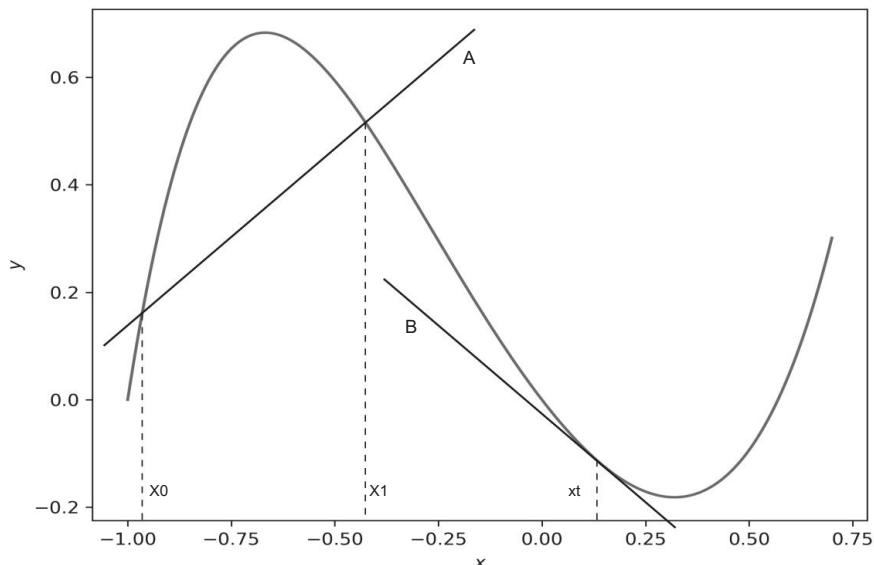


Figura 7-1: Una curva con una recta secante (A) y una recta tangente (B)

La recta denominada A cruza la curva en dos puntos,  $x_1$  y  $x_0$ . Una recta que pasa entre dos puntos de una curva se llama recta secante. La otra recta, B, apenas toca la curva en el punto  $x_t$ . Las rectas que tocan una curva en un punto se llaman rectas tangentes. Volveremos a las rectas secantes en la siguiente sección, pero, por ahora, observe que la recta tangente tiene una pendiente particular en  $x_t$  y que una recta secante se convierte en una recta tangente a medida que aumenta la distancia entre los puntos  $x_1$  y  $x_0$  hacia cero.

Imaginemos que movemos el punto  $x_t$  de un lugar a otro a lo largo de la curva; podemos ver que la pendiente de la recta tangente en  $x_t$  cambiaría con ella. A medida que nos acercamos al punto mínimo de la curva, cerca de  $x = 0,3$ , vemos que la pendiente se vuelve cada vez menos profunda. Si nos acercamos por la izquierda, la pendiente es negativa y cada vez es menos negativa. Si nos acercamos por la derecha, la pendiente es positiva pero cada vez es menor. En el punto mínimo real, cerca de  $x = 0,3$ , la recta tangente es horizontal, con una pendiente igual a cero. De manera similar, si nos acercamos al máximo de la curva, cerca de  $x = -0,8$ , la pendiente también se acerca a cero.

Podemos ver que la recta tangente nos dice cómo cambia la curva en un punto. Como veremos más adelante en el capítulo, el hecho de que la pendiente de una recta tangente sea cero en los mínimos y máximos de una curva nos indica un método para encontrar estos puntos. Los puntos donde la pendiente de la recta tangente es cero se conocen como puntos estacionarios.

Por supuesto, para aprovechar la pendiente de la recta tangente, debemos poder encontrar su valor para cualquier  $x$  en la curva. La siguiente sección nos mostrará cómo.

## Derivados

La sección anterior introdujo la idea de rectas secantes y tangentes e insinuó que conocer la pendiente de la recta tangente en cualquier punto de una curva es algo potencialmente útil. La pendiente de la recta tangente en un punto  $x$  se conoce como derivada en  $x$ . Nos dice cómo cambia la curva (función) en el punto  $x$ , es decir, cómo cambia el valor de la función con un cambio infinitesimal en  $x$ . En esta sección, definiremos formalmente la derivada y aprenderemos reglas abreviadas para calcular derivadas de funciones de una sola variable,  $x$ .

### Una definición formal

Un curso típico de cálculo del primer semestre le presenta las derivadas mediante el estudio de los límites. Mencioné anteriormente cómo la pendiente de la línea secante entre dos puntos en una curva se convierte en una línea tangente cuando los puntos colapsan uno encima del otro, y ese es un lugar donde los límites entran en juego.

Por ejemplo, si  $y = f(x)$  es una curva y tenemos dos puntos en la curva, entonces la pendiente,  $\Delta y / \Delta x$ , entre estos puntos es  $x_0$  y  $x_1$ ,

$$\frac{\Delta y}{\Delta x} = \frac{y_1 - y_0}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad (7.1)$$

Este es el aumento sobre la carrera que quizás recuerdes haber aprendido en la escuela. El aumento,  $\Delta y = y_1 - y_0 = f(x_1) - f(x_0)$ , se divide por la carrera,  $\Delta x = x_1 - x_0$ . Normalmente usamos  $\Delta$  como prefijo para indicar el cambio en alguna variable.

Si definimos  $h = x_1 - x_0$ , podemos reescribir la ecuación 7.1 como

$$\frac{\Delta y}{\Delta x} = \frac{f(x_0 + h) - f(x_0)}{h}$$

ya que  $x_1 = x_0 + h$ .

En esta nueva forma, podemos encontrar la pendiente de la recta tangente en  $x_0$  dejando que  $h$  se acerque cada vez más a cero,  $h \rightarrow 0$ . Dejar que un valor se acerque a otro valor es un límite. Dejando que  $h \rightarrow 0$  mueva los dos puntos, estamos calculando la pendiente entre cada vez más cerca. Esto lleva directamente a la definición de la derivada.

$$= f'(x) \equiv \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (7.2)$$

donde  $dy/dx$  o  $f'(x)$  se utiliza para representar la derivada de  $f(x)$ .

Antes de entrar en lo que significa la derivada, tomemos un minuto para analizar la notación. El uso de  $f'(x)$  para la derivada sigue a Joseph-Louis Lagrange.

Leibniz usó  $dy/dx$  para reflejar la notación de pendiente con  $\Delta \rightarrow d$ . Si  $\Delta y$  es un cambio en  $y$  entre dos puntos,  $dy$  es el cambio infinitesimal en  $y$  en un solo punto. Newton usó otra notación,  $f'$ , donde el punto representa la derivada de  $f$ . Los físicos suelen utilizar la notación de Newton para el caso específico de las derivadas con respecto al tiempo. Por ejemplo, si  $f(t)$  es la posición de una part-f(t) es la derivada con respecto a  $t$ , es decir, cambia la posición en el tiempo. La  $f'$  en función del tiempo,  $t$ , entonces cómo forma en que una posición cambia en el tiempo es la velocidad (velocidad si se usan vectores). Verás todas estas notaciones en los libros.  $'(X)$

Mi preferencia es preservar  $f$  para funciones de tiempo y usar  $f'$  de Lagrange y  $dy/dx$  de Leibniz indistintamente en otros lugares.

Aunque la ecuación 7.2 anterior es bastante tediosa y la pesadilla de muchos estudiantes principiantes de cálculo, al menos hasta que lleguen a la integración, podrías trabajar con ella si fuera necesario. Sin embargo, no hablaremos de la integración en absoluto en este libro, por lo que puede respirar profundamente y relajarse.

Después de la lucha con los límites, los estudiantes de cálculo descubren un secreto: un pequeño conjunto de reglas les permitirá calcular prácticamente todas las derivadas sin utilizar límites. Comenzaremos presentando estas reglas, una a la vez con ejemplos, y luego, al final de esta sección, las reuniremos todas en una forma adecuada para una camiseta.

Sin embargo, antes de profundizar en las reglas, dediquemos un poco más de tiempo a analizar lo que nos dice la derivada. Arriba, mencioné que la derivada da cómo cambia una posición en el tiempo. Esto es cierto para todos los derivados; nos dicen cómo algo está cambiando con respecto a cómo algo más está cambiando. Incluso vemos esto en la notación de Leibniz,  $dy/dx$ , cómo  $dy$  cambia ante un cambio en  $dx$ . La derivada en  $x$  nos dice cómo es la función

cambiando en x. Como veremos, la derivada de  $f(x)$  es en sí misma una nueva función de x. Si elegimos un  $x_0$  específico , entonces sabemos que  $f'(x_0)$  es el valor de la función en  $x_0$  .

De manera similar, si conocemos la derivada, entonces  $f'(x_0)$  es la rapidez con la que, y en qué dirección cambia la función  $f(x)$  en  $x_0$ . Considera la definición de velocidad como cómo cambia la posición con el tiempo. Incluso lo decimos con palabras: mi velocidad actual es de 30 mph (millas por hora), un cambio de posición con respecto a un cambio de tiempo.

Usaremos derivados como las tasas y veremos cómo cambiar una cosa afecta a otra. Al final, para el aprendizaje profundo, queremos saber cómo cambiar el valor de un parámetro en una red cambiará en última instancia la función de pérdida, el error entre lo que la red debería generar y lo que realmente genera. '(x) es función de x, entonces

deberíamos poder tomar la derivada. Si  $f'(x)$  es la primera derivada. Su derivada, que Leibniz denotamos como  $f''(x)$ , de ella. Llamamos  $f''$  es la segunda derivada. En la notación de escribimos d  $\frac{d^2y}{dx^2}$ . La segunda derivada nos dice cómo cambia la primera derivada con respecto a  $x$ . La física ayuda aquí. La primera derivada de la posición en función del tiempo ( $f$ ) es la velocidad,  $f'$ : cómo cambia la posición con el tiempo. Por lo tanto, la segunda derivada de la posición,  $f''$ , que es la primera derivada de la velocidad, es cómo la velocidad cambia con el tiempo. A esto lo llamamos aceleración.

En teoría, la cantidad de derivadas que podemos calcular es infinita. En realidad, muchas funciones terminan en última instancia con una derivada que es un valor constante. Como un valor constante no cambia, su derivada es cero.

En resumen, la derivada de  $f(x)$  es otra función,  $f'(x)$  o  $dy/dx$ , que nos dice la pendiente de la recta tangente a  $f(x)$  en cada punto. Y, dado que  $f'(x)$  es  $2y/dx^2$ , una función de  $x$ , tiene una derivada,  $f''(x)$  o  $d^2y/dx^2$ . La segunda derivada, que nos ~~dice~~  $\rightarrow$  si  $f'(x)$  cambia en cada  $x$ , y así sucesivamente. Veremos a continuación cómo hacer uso de la primera y segunda derivada. Por ahora, aprendamos las reglas de diferenciación, el acto de calcular una derivada.

## Reglas básicas

Mencionamos una regla en la sección anterior: que la derivada de una constante,  $c$ , es cero. Entonces, escribimos

$$\frac{dC}{dx} = 0$$

donde usamos la notación de Leibniz en forma de operador:  $d/dx$ . Piense en  $d/dx$  como algo que opera en lo que sigue; lo hace de la misma manera que lo hace la negación: para negar  $c$ , escribimos  $-c$ ; para diferenciar  $c$ , escribimos que la presión no tiene  $x$ ,  $\frac{dc}{dx}$ . Si nuestro ex-  
luego la trataremos como una constante y la derivada será cero.

La regla del poder

La derivada de una potencia de x usa la regla de la potencia,

$$\frac{d}{dx} x^n = n x^{n-1}$$

donde a es una constante y n es un exponente que no necesita ser un número entero.

Veamos algunos ejemplos:

$$\frac{d}{dx} x^3 = 3x^2$$

$$\frac{d}{dx} 4x^2 = (2)4x^{2-1} = 8x$$

$$\frac{d}{dx} x = (1)x^{1-1} = (1)x^0 = 1$$

$$\frac{d}{dx} \sqrt{x} = \frac{d}{dx} x^{\frac{1}{2}} = \frac{1}{2} x^{-\frac{1}{2}} = \frac{1}{2\sqrt{x}}$$

$$\frac{d}{dx} 0,07 = 0,007x^{-0,93} = \frac{0,007}{x^{0,93}}$$

A menudo construimos expresiones algebraicas a partir de términos que se suman y restan. La diferenciación es un operador lineal, por lo que podemos escribir

$$\frac{d}{dx} (f(x) \pm g(x)) = f(x) \pm g'(x)$$

Esto significa que calculamos las derivadas término por término. Por ejemplo, con el conjunto de reglas que tenemos hasta ahora, ahora sabemos cómo calcular la derivada de un polinomio:

$$\frac{d}{dx} (3x^4 - 2x^3 + 3x + 4) = 12x^3 - 6x^2 + 3$$

$$\frac{d}{dx} (x^5 - 7x^2 + 42) = 5x^4 - 14x$$

En general, entonces,

$$\frac{d}{dx} (ax^n + bx^{n-1} + cx^{n-2} + \dots + yx + z) = anx^{n-1} + b(n-1)x^{n-2} +$$

$$c(n-2)x^{n-3} + \dots + y$$

donde vemos que la derivada de un polinomio de grado  $n$  es otro polinomio de grado  $n - 1$  y que cualquier término constante en el polinomio original cae a cero.

La regla del producto

La diferenciación de funciones multiplicadas tiene su propia regla, la regla del producto:

$$\frac{d}{dx} f(x)g(x) = f'(x)g(x) + f(x)g'(x)$$

La derivada del producto es la derivada de la primera función por la segunda más la derivada de la segunda por la primera. Considere los siguientes ejemplos:

$$\frac{d}{dx} (x^2 - 4)(3x + 3) = (2x)(3x + 3) + (x^2 - 4)(3)$$

$$2 = 9x + 6x - 12$$

$$-d(x - 3)(4x + 5) = (1)(4x + 5) + (x - 3)(4) dx$$

$$= 8x - 7$$

$$\frac{d}{dx} (2x + 2)(x^2 - x - 3) = (2)(x^2 - x - 3) + (2x + 2)(2x - 1)$$

$$2 = 6x - 8$$

La regla del cociente

La derivada de una función dividida por otra función sigue la regla del cociente:

$$\frac{d}{dx} \left( \frac{f(x)}{g(x)} \right) = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}$$

Esto lleva a ejemplos como estos:

$$\begin{aligned} \frac{d}{dx} \left( \frac{5x-3}{2x+1} \right) &= \frac{5(2x+1) - (5x-3)(2)(2x)}{(2x+1)^2} \\ &= \frac{11}{(2x+1)^2} \\ \frac{d}{dx(x^3-9)} \frac{x^2+2x-3}{x^3-9} &= \frac{(2x+2)(x^3-9) - (x^2+2x-3)(3x^2)}{(x^3-9)^2} \\ &= \frac{4x^3+2x^2+4x-9x}{(x^3-9)^2} \end{aligned}$$

#### La regla de la cadena

La siguiente regla se refiere a la composición de funciones. Dos funciones se componen cuando la salida de una se utiliza como entrada para otra. La regla de la cadena se aplica a las composiciones de funciones y es de fundamental importancia en el entrenamiento de redes neuronales. La regla es

$$df(g(x)) = f'(g(x))g'(x) dx$$

Multiplicamos la derivada de la función externa, usando  $g(x)$  como variable, por la derivada de la función interna con respecto a  $x$ .

Como primer ejemplo, considere la función  $f(x) = (x^2 + 2x + 3)^2$ . ¿Es esto la composición de dos funciones? Es. Definamos  $f(g) = g^2$  y  $g(x) = x^2 + 2x + 3$ . Entonces, podemos encontrar  $f(x)$  reemplazando cada instancia de  $g$  en  $f(g)$  con la definición de  $g$  en términos de  $x$ :  $g(x) = x^2 + 2x + 3$ . Esto da

$$f(x) = g(x)^2 = (x^2 + 2x + 3)^2$$

que es, naturalmente, con lo que empezamos. Para encontrar  $f'(x)$ , primero encontramos  $f'(g)$ , la derivada de  $f$  con respecto a  $g$ , y luego multiplicamos por  $g'(x)$ , la derivada de  $g$  con respecto a  $x$ . Como paso final, reemplazamos las referencias a  $g$  con su definición en términos de  $x$ . Entonces, calculamos  $f'(x)$  como

$$\begin{aligned} f'(x) &= (2g)(2x+2)f \\ &= 2(x^2 + 2x + 3)(2x+2) \\ &= 4(x+1)(x^2 + 2x + 3) \end{aligned}$$

Por lo general, no mencionamos explícitamente  $f(g)$  y  $g(x)$ , pero mentalmente trabajamos en el mismo proceso. Veamos algunos ejemplos más. En este queremos encontrar

$$\frac{d}{dx} (2(4x - 5)^2 + 3)$$

Si usamos la regla de la cadena, vemos que tenemos  $f(g) = 2g^2 + 3$  y  $g(x) = 4x - 5$ . Por lo tanto, podemos escribir

$$\begin{aligned}\frac{d}{dx} (2(4x - 5)^2 + 3) &= (4g)(4) \\ &= 4(4x - 5)(4) \\ &= 16(4x - 5)\end{aligned}$$

donde  $f'(g) = 2g$  y  $g'(x) = 4$ . Con un poco de práctica, imaginariamos mentalmente el  $4x - 5$  de  $f(x)$  como su propia variable (la sustitución de  $g$ ) y luego recordaríamos para multiplicar la derivada de  $4x - 5$  cuando hayas terminado. ¿Y si no viéramos la composición? ¿Qué pasaría si expandiéramos toda la función,  $f(x)$ , y luego tomáramos la derivada? Será mejor que obtengamos la respuesta que encontramos anteriormente usando la regla de la cadena. Vamos a ver . . .

$$\begin{aligned}f(x) &= 2(4x - 5)^2 + 3 \\ &= 2(16x^2 - 40x + 25) + 3 \\ &= 32x^2 - 80x + 53 \\ f'(x) &= 64x - 80 \\ &= 16(4x - 5)\end{aligned}$$

Esto es lo que encontramos arriba, por lo que nuestra aplicación de la regla de la cadena es correcta.

Veamos otro ejemplo. Si  $f(x) = \frac{u(x)}{v(x)}$ , ¿Cómo deberíamos pensar en calcular la derivada? Si miramos la función como  $f(x) = u(x) - v(x)$ , con  $u(x) = x^2$  y  $v(x) = 3x^2$ , podemos usar la regla del cociente y obtener lo siguiente.

$$\begin{aligned}
 f'(x) &= \frac{'tú v - uv'}{2v} \\
 &= \frac{0(3x^2) - 1(6x)}{(3x^2) 2} \\
 &= \frac{-6x}{9x^4} \\
 &= -\frac{2}{3x^3}
 \end{aligned}$$

Aquí utilizamos una notación abreviada con  $u$  y  $v$  que elimina la función formal de la notación  $x$ .

También podemos imaginar  $f(x)$  como  $(3x^2)^{-1}$ . Si lo pensamos de esta manera, podemos aproximarnos y aplicar la regla de la cadena y la regla de la potencia para obtener

$$\begin{aligned}
 f'(x) &= (-1)(3x^2)^{-2}(-2(6x)) \\
 &= \frac{-6x^9x}{4} \\
 &= -\frac{2}{3x^3}
 \end{aligned}$$

demostrando que a veces hay más de una forma de calcular una derivada.

Presentamos la regla de la cadena usando la notación de Lagrange. Más adelante en el capítulo lo veremos nuevamente usando la notación de Leibniz. Sigamos ahora y presentemos un conjunto de reglas para funciones trigonométricas.

### Reglas para funciones trigonométricas

Las derivadas de las funciones trigonométricas básicas son sencillas:

$$\frac{d}{dx} \sin x = \cos x$$

$$-\frac{d}{dx} \cos x = -\sin x$$

$$-\frac{d}{dx} \tan x = \sec^2 x$$

Podemos ver que la última regla es correcta si aplicamos las reglas básicas de diferenciación a la definición de la tangente:

$$\begin{aligned}
 & \frac{d}{dx} \operatorname{bronceado} x = \frac{d}{dx} \left( \frac{\operatorname{pecado} x}{\cos x} \right) \\
 &= \frac{\cos x \cos x - \operatorname{sen} x(-\operatorname{sen} x)}{\cos^2 x} \\
 &= \frac{\operatorname{sen} 2x + \cos 2x}{\cos^2 x} \\
 &= \frac{1}{\cos^2 x} \\
 &= \operatorname{seg} 2x
 \end{aligned}$$

Recuerda que  $\operatorname{sec} x = 1/\cos x$  y  $\operatorname{sen} 2x + \cos 2x = 1$ .

Veamos algunos ejemplos usando las nuevas reglas trigonométricas. Empezaremos con uno que compone una función con una función trigonométrica:

$$\begin{aligned}
 \frac{d}{dx} \operatorname{pecado}(x^3 - 3x) &= \cos(x^3 - 3x)(3x^2 - 3) \\
 &= 3(x^2 - 1) \cos(x^3 - 3x)
 \end{aligned}$$

Podemos ver que esta es una composición de  $f(g) = \sin(g)$  y  $g(x) = x^3 - 3x$ , por<sup>3</sup> sabemos que podemos aplicar la regla de la cadena para obtener la derivada como  $f' \circ g'(x)$  donde  $f'(x) = \cos(x)$  y  $g'(x) = 3x^2 - 3$ . La respuesta.

Veamos una composición más complicada:

$$\begin{aligned}
 \frac{d}{dx} \operatorname{pecado}^2(x^3 - 3x) &= 2 \operatorname{pecado}(x^3 - 3x) \cos(x^3 - 3x)(3x^2 - 3) \\
 &= 6(x^2 - 1) \operatorname{pecado}(x^3 - 3x) \cos(x^3 - 3x)
 \end{aligned}$$

Esta vez, dividimos la composición como  $f(g) = g$ . Sin embargo,  $g(x) = \operatorname{pecado}(x^3 - 3x)$ ,  $g(x)$  es en sí misma una composición de  $g(u) = \sin(u)$  y  $u(x) = x^3 - 3x$  como ejemplo anterior. Entonces, el primer paso es escribir lo siguiente.

$$\begin{aligned}
 f'(x) &= f'(g(x))g'(x) \\
 &= 2g(x)g'(x) \\
 &= 2 \operatorname{sen}(x^3 - 3x)g'(x)
 \end{aligned}$$

La primera línea es la definición de la derivada de una composición. La segunda línea sustituye la derivada de  $f(g)$ , que es  $2g$ , y la tercera línea reemplaza  $g(x)$  con  $\operatorname{sen}(x - 3x)$ . Ahora, sólo necesitamos encontrar  $g'(x)$ , lo cual podemos hacer usando la regla de la cadena por segunda vez con  $g(u) = \operatorname{sen} u$  y  $u(x) = x - 3x$ , ya que  $g'(x) = \cos(x - 3)$ , lo hizo en el ejemplo anterior. Hacer esto nos da  $g'(x) = \cos(x - 3)$ .

$$\begin{aligned}
 f'(x) &= 2 \operatorname{sen}(x^3 - 3x)\cos(x^3 - 3x)(3x^2 - 3) \\
 &= 2 \operatorname{sen}(x^3 - 3x)\cos(x^3 - 3x)(3x^2 - 3) \\
 &= 6(x^2 - 1)\operatorname{sen}(x^3 - 3x)\cos(x^3 - 3x)
 \end{aligned}$$

Hagamos un ejemplo más. Este involucrará más de una función trigonométrica. Queremos ver cómo calcular

$$\frac{d}{dx} (\operatorname{sen}^3(x)) = 2\operatorname{sen}^2(x)\cos(x)$$

Como suele ocurrir cuando se trabaja con funciones trigonométricas, las identidades surgen en juego. Aquí vemos que  $f(x)$  se puede reescribir:

$$\begin{aligned}
 f(x) &= \frac{\operatorname{sen}^3(x)}{\cos(x)} \\
 &= \operatorname{sen}^2(x) \overline{(\operatorname{sen}(x))} \\
 &= \operatorname{sen}^2(x)\tan(x)
 \end{aligned}$$

Ahora la derivada usa las reglas trigonométricas, la definición de la secante, la regla de la cadena y la regla del producto, como se muestra a continuación.

$$f'_d(x) = \frac{d}{dx} (\operatorname{pecado2}(x)) \tan(x) + \operatorname{pecado2}(x) \frac{d}{dx} (\operatorname{broncado}(x))$$

$$= 2 \operatorname{sen}(x) \cos(x) \tan(x) + \operatorname{sen}^2(x) \sec^2(x)$$

$$= 2 \sin(x) \cos(x) \tan(x) + \sin^2(1) \cos^2(x) = 2 \sin(x) \cos(x)$$

$$(\sin(x) \cos(x)) + \tan^2(x)$$

$$= 2 \operatorname{sen}^2(x) + \operatorname{tan}^2(x)$$

Sigamos adelante y veamos las derivadas de exponentiales y logaritmos.

## Reglas para exponentiales y logaritmos

La derivada de  $e$  ( $e \approx 2,718\dots$ ), donde  $e$  es la base del logaritmo natural, es particularmente simple. Es él mismo:

$$\frac{d}{e dx} x = mi x$$

Cuando el argumento es función de x, esto se convierte en

$$\frac{d}{dx} m_i \text{gramo}(x) = \text{gramo}'(x) e g(x) \quad (7.3)$$

Si la derivada de  $e^x$  es  $e^x$ , ¿Cuál es la derivada de  $a^x$  cuando  $a$  es un número real? distinta de  $e$ ? Para ver la respuesta, debemos recordar que  $e^x$  y  $\ln x = a$ . Entonces, el logaritmo natural que usa base  $e$ , son funciones inversas, por lo que podemos escribir

$$a^x \quad x = (\ln a) \quad a = e^{\ln a}$$

cómo encontrar la derivada de  $e$ .

x ln a de la ecuación 7.3 anterior. Sabemos

$$\frac{d}{dx} e^{x \ln a} = \ln(a)e^{x \ln a}$$

pero  $\text{ex} \ln a = a$      $\times$     entonces tenemos

$$\frac{d}{dx} x = \text{unx en un}$$

y en general.

$$\frac{d}{dx} g(x) \cdot (x) a^g(x) = \ln(a) g$$
(7.4)

Observe que si  $a = e$ , tenemos  $\ln(e) = 1$  y la ecuación 7.4 se convierte en la ecuación 7.3.

Veamos ahora la derivada del propio tronco natural. Es

$$\frac{d}{dx} \ln x = \frac{1}{x}$$

Cuando el argumento es función de  $x$ , esto se convierte en

$$\frac{d}{dx} \ln(g(x)) = \frac{g'(x)}{g(x)} \quad (7.5)$$

Quizás te estés preguntando: ¿Cómo encontramos la derivada de un logaritmo que usa una base distinta a  $e$ ? Por ejemplo, ¿cuál es la derivada de  $\log_{10} x$ ? Para responder a esta pregunta, haremos algo similar a lo que hicimos anteriormente para la derivada de  $a$  el  $x^a$ . Escribimos el logaritmo de  $x$  para alguna base,  $b$ , en términos de registro natural, como

$$\log x = \frac{\ln x}{\ln b}$$

Aquí,  $\ln b$  es una constante que no depende de  $x$ . Además, ahora sabemos cómo encontrar la derivada de  $\ln x$ , por lo que vemos que la derivada de  $\log_b x$  debe ser

$$\frac{d}{dx} \log x = \frac{1}{x \ln b}$$

para cualquier número real base  $b \neq 1$ . Y, aún más en general,

$$\frac{d}{dx} \log_b g(x) = \frac{g'(x)}{g(x) \ln b} \quad (7.6)$$

donde, nuevamente, observamos que si  $b = e$ , obtenemos  $\ln e = 1$ , y la ecuación 7.6 se convierte en la ecuación 7.5.

Con la ecuación 7.6, llegamos al final de nuestras reglas para derivados. Juntémoslos ahora en una sola tabla a la que podamos consultar durante el resto del libro. El resultado es la Tabla 7-1.

Ahora sabemos cómo encontrar derivadas. Le animo a que busque hojas de práctica con respuestas elaboradas para convencerse de que comprende las reglas y cómo aplicarlas. Sigamos adelante y veamos cómo podemos usar derivadas para encontrar los mínimos y máximos de funciones. Encontrar mínimos es fundamental para el entrenamiento de redes neuronales.

Tabla 7-1: Las reglas de diferenciación

Tipo	Regla
Constantes	$\frac{d}{dx} c = 0$
Potestades	$\frac{d}{dx} ax^n = anx^{n-1}$
sumas	$\frac{d}{dx} f(x) \pm g(x) = f'(x) \pm g'(x)$
Productos	$\frac{d}{dx} f(x)g(x) = f'(x)g(x) + f(x)g'(x)$
Cocientes	$\frac{d}{dx} \frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}$
Cadena	$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$
Trigonometría	$\frac{d}{dx} \text{sen}(g(x)) = g'(x) \cos(g(x))$ $\frac{d}{dx} \cos(g(x)) = -g'(x) \text{sen}(g(x))$ $\frac{d}{dx} \tan(g(x)) = g'(x) \sec^2(g(x))$
Exponentes	$\frac{d}{dx} a^{\text{grado}(x)} = \text{grado}'(x) a^{\text{grado}(x)}$ $\frac{d}{dx} a^g = \ln(a)g a^{g-1}$
Logaritmos	$\frac{d}{dx} \ln g(x) = \frac{g'(x)}{g(x)}$ $\frac{d}{dx} \log_b g(x) = \frac{g'(x)}{g(x) \ln b}$

## Mínimos y máximos de funciones

Anteriormente, definí los puntos estacionarios como lugares donde la primera derivada de una función es cero, es decir, lugares donde la pendiente de la recta tangente es cero.

Podemos usar esta información para decidir si un punto particular, llámelo  $x_m$ , es un mínimo o máximo de la función,  $f(x)$ . Si  $x_m$  es un mínimo, es un nivel bajo. punto de la función, donde  $f(x_m)$  es menor que cualquier punto inmediato izquierda o derecha de  $f(x_m)$ . De manera similar, si  $f(x_m)$  es mayor que cualquier punto inmediatamente a la izquierda o a la derecha,  $f(x_m)$  es un máximo. Nos referimos colectivamente a mínimos y máximos como extremos de  $f(x)$  (singular, extremo).

En términos de la derivada, un mínimo es un lugar donde la derivada de los puntos inmediatamente a la izquierda de  $x_m$  es negativa y la derivada de los puntos directamente a la derecha de  $x_m$  es positiva. Los máximos son al revés: las derivadas a la izquierda son positivas y las derivadas a la derecha de  $x_m$  son negativas.

Vuelva a mirar la Figura 7-1. Allí, tenemos un máximo local en aproximadamente  $x = -0,8$  y un mínimo local en aproximadamente  $x = 0,3$ . Digamos que el máximo está en realidad en  $x_m = -0,8$ . Este es un máximo porque si miramos cualquier punto  $x_p$  cercano a  $x_m$ ,  $f(x_p)$  es menor que  $f(x_m)$ . Del mismo modo, si el mínimo está en  $x_m = 0,3$ , eso se debe a que cualquier punto  $x_p$  cercano tiene  $f(x_p) > f(x_m)$ . Si imaginamos la recta tangente deslizándose a lo largo de la gráfica, a medida que se aproxima a  $x = -0,8$ , vemos que la pendiente es positiva pero se dirige hacia cero. Si pasamos  $x = -0,8$ , la pendiente ahora es negativa. Lo contrario es cierto para el mínimo en  $x = 0,3$ . Las rectas tangentes que se acercan por la izquierda tienen pendiente negativa, pero una vez que pasan  $x = 0,3$ , la pendiente es positiva.

Leerás y escucharás los términos global y local aplicados a mínimos y máximos. El mínimo global de  $f(x)$  es el más bajo de todos los mínimos de  $f(x)$ , y el máximo global es el más alto de todos los máximos. Otros mínimos y máximos, entonces, se consideran locales; son efectivos en una región particular, pero hay otros mínimos que son más bajos o máximos que son más altos.

Debemos tener en cuenta que no todas las funciones tienen mínimos o máximos. Por ejemplo, una recta,  $f(x) = mx + b$ , no tiene mínimos ni máximos, porque no hay puntos en la recta que satisfagan los requisitos de ninguno de los dos.

Entonces, si la primera derivada,  $f'(x)$ , es cero, tenemos un mínimo o un máximo, ¿verdad? No tan rápido. En otros puntos estacionarios, la primera derivada puede ser cero, pero no se cumplen los criterios restantes para un mínimo o un máximo. Estos puntos a menudo se denominan puntos de inflexión o, si están en múltiples dimensiones, puntos de silla. Por ejemplo,  $y = x^3$ . La primera derivada es  $y' = 3x^2$ , considere  $y = x$  y la segunda derivada es  $y'' = 6x$ . Tanto la primera como la segunda derivada son cero en  $x = 0$ . Sin embargo, como podemos ver en la figura 7-2, la pendiente es positiva tanto hacia la izquierda como hacia la derecha inmediata de  $x = 0$ . Por lo tanto, la pendiente nunca cambia de positiva a negativo o de negativo a positivo, lo que significa que  $x = 0$  no es un extremo sino un punto de inflexión.

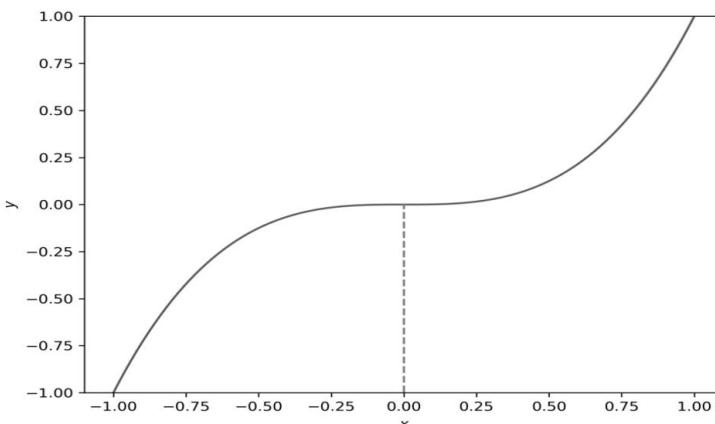


Figura 7-2: Una gráfica de  $y = x^3$  mostrando un punto de inflexión en  $x = 0$

Ahora supongamos que  $x_s$  es un punto estacionario de modo que  $f'(x_s) = 0$ . Si elegimos otros dos puntos,  $x_{s-}$  y  $x_{s+}$ , uno justo a la izquierda de  $x_s$  y el otro justo a la derecha, para algunos puntos muy pequeños ( $\epsilon$ ), tenemos cuatro posibilidades para los valores de  $f'(x_{s-})$  y  $f'(x_{s+})$ , que se muestran en la Tabla 7-2.

Tabla 7-2: Identificación de puntos estacionarios

Signo de $f'(x_{s-})$ , $f'(x_{s+})$	Tipo de punto estacionario en $x_{s-} < x_s < x_{s+}$
+,-	Máximo
-,+	Mínimo
+,+	Ni
-,-	Ni

Por lo tanto, el valor de la primera derivada en un punto estacionario candidato no es suficiente para decirnos si el punto representa un mínimo o un máximo. Podemos mirar la región alrededor del punto candidato para ayudarnos a decidir. También podemos fijarnos en el valor de  $f''(x)$ , la segunda derivada de  $f(x)$ . Si  $x_s$  es un punto estacionario donde  $f'(x_s) = 0$ , el signo de  $f''(x_s)$  puede indicarnos qué tipo de punto estacionario podría ser  $x_s$ . Si  $f''(x_s) < 0$ , entonces  $x_s$  es un máximo de  $f(x)$ . Si  $f''(x_s) > 0$ ,  $x_s$  es un mínimo. Si  $f''(x_s) = 0$ , la segunda derivada no es útil; necesitaremos probar explícitamente puntos cercanos con la primera derivada.

¿Cómo encontramos puntos estacionarios candidatos en primer lugar? Para funciones algebraicas, resolvemos  $f'(x) = 0$ ; encontramos el conjunto solución de todos los valores de  $x$  que hacen que la primera derivada de  $f(x)$  sea cero. Luego usamos las pruebas de derivadas para decidir cuáles son mínimos, máximos o puntos de inflexión.

Para muchas funciones, podemos encontrar las soluciones de  $f'(x) = 0$  directamente. Por ejemplo, si  $f(x) = x^3 - 2x + 4$ , tenemos  $f'(x) = 3x^2 - 2$ . Si igualamos esto a  $3x^2 - 2 = 0$  y lo resolvemos usando la fórmula cuadrática, encontramos que es cero,  $3x^2 = 2$ . Los puntos estacionarios:  $x_0 = -\sqrt{6}/3$  y  $x_1 = \sqrt{6}/3$ . La segunda derivada de  $f(x)$  es  $f''(x) = 6x$ . El signo de  $f''(x_0)$  es negativo; por lo tanto,  $x_0$  representa un máximo. Y como el signo de  $f''(x_1)$  es positivo,  $x_1$  es un mínimo.

Podemos ver que las pruebas de derivadas son correctas. La parte superior de la nos muestra una gráfica de  $f(x) = x^3 - 2x^2 + 4$ . Figura 7-3 –  $2x^2 + 4$ , donde  $x_0$  es un máximo y  $x_1$  es un mínimo.

Veamos un ejemplo más. Esta vez, tenemos  $f(x) = x^5 - 2x^3 + x^2 + 2$ , el gráfico inferior de la Figura 7-3. Encontramos la primera derivada y la ponemos a cero:

$$f'(x) = 5x^4 - 6x^2 + 1 = 0$$

Si sustituimos  $u = x^2$ , podemos resolver las raíces de  $f'(x) = 0$  al encontrar raíces de  $5u^2 - 6u + 1 = 0$  y establecerlas iguales a  $x$  y  $x = \pm\sqrt{\frac{1}{5}}$ . Hacer esto nos da  $u = 1$ ,  $\pm\sqrt{\frac{1}{5}}$ , entonces tenemos cuatro puntos estacionarios. Para probarlos podemos usar la prueba de la segunda derivada. La segunda derivada es  $f''(x) = 20x^3 - 12x$ .

Sustituyendo los puntos estacionarios en  $f''$  da

$$f''(x_0 = -1) = -8$$

$$f''(x_1 = ) \approx -3.5777 \sqrt{5}$$

$$f''(x_2 = ) \approx -3.5777 \sqrt{5}$$

$$f''(x_3 = 1) = 8$$

lo que significa que  $x_0$  es un máximo,  $x_1$  es un mínimo,  $x_2$  es otro máximo y  $x_3$  es un mínimo. La Figura 7-3 confirma nuevamente nuestras conclusiones.

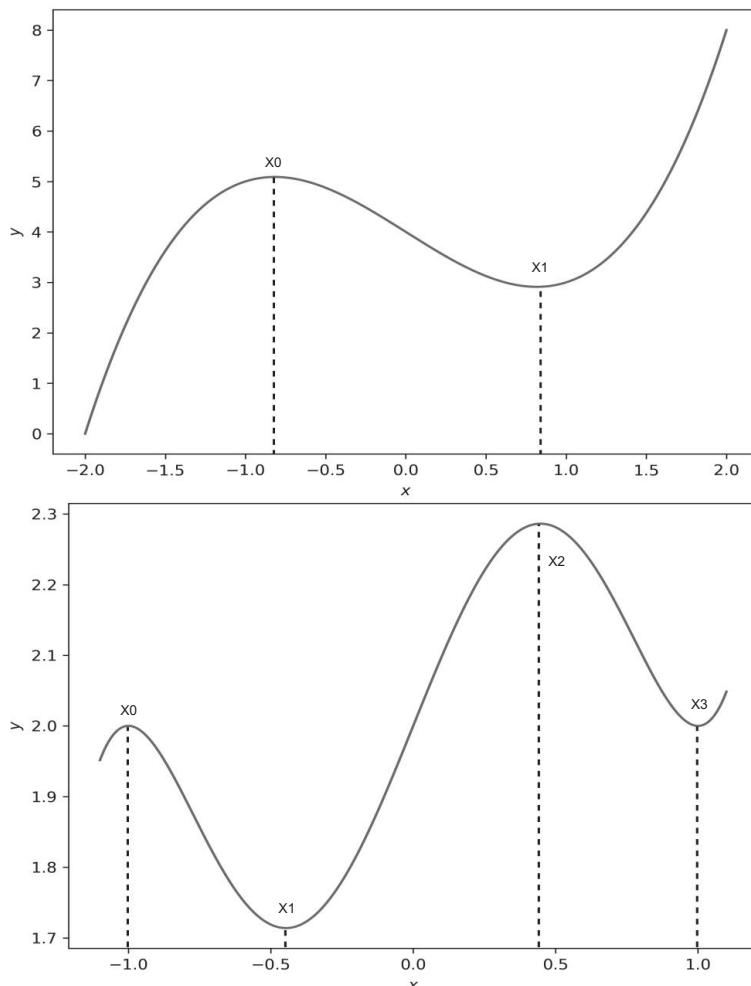


Figura 7-3: Gráficas de  $f(x) = \frac{5}{3} - 2x^3 + x^2 + 2$  (arriba) y  $x$  con extremos marcados

¿Qué pasa si no podemos encontrar fácilmente los puntos estacionarios de una función? Tal vez no podemos resolver la función algebraicamente, o tal vez no se pueda expresar en forma cerrada, lo que significa que ningún conjunto finito de operaciones la representa. Un curso de cálculo típico no está interesado en estas situaciones. Aún así, debemos serlo, porque una forma de pensar en una red neuronal es como un aproximador de funciones, una función cuya función no se puede expresar directamente. ¿Podemos todavía utilizar de forma rentable nuestros nuevos conocimientos sobre derivados? La respuesta es sí, podemos. Podemos utilizar la derivada como indicador para decirnos cómo acercarnos cada vez más al extremo. Esto es lo que hace el descenso de gradiente, y dedicaremos bastante tiempo a analizarlo más adelante en el libro.

Por ahora, sin embargo, sigamos adelante y examinemos funciones de más de una variable y ver qué efecto tiene esto en la idea de una derivada.

## Derivadas parciales

Hasta ahora nos hemos centrado exclusivamente en funciones de una variable,  $x$ . ¿Qué sucede con la noción de diferenciación cuando tenemos funciones de más de una variable, digamos,  $f(x,y)$  o  $f(x_0, x_1, x_2, \dots, x_n)$ ? Para manejar estos casos, introduciremos la idea de derivada parcial. Tenga en cuenta que, para mayor claridad, usaremos la notación de Leibniz en esta sección.

La ecuación 7.2 definió la derivada de  $f(x)$  con respecto a  $x$ . Si  $x$  es la única variable, ¿por qué agregamos la frase adicional "con respecto a  $x$ "? Ahora descubriremos por qué: la derivada parcial con respecto a una de las variables de la expresión se encuentra manteniendo fijas todas las demás variables. Los tratamos como si fueran constantes. Luego decimos que estamos calculando la derivada parcial con respecto a la que no se mantiene fija.

Veamos un ejemplo. Sea  $f(x,y) = xy + x/y$ . Luego, podemos calcular dos derivadas parciales, una con respecto a  $x$  y otra con respecto a  $y$ :

$$\frac{\partial f}{\partial x} = y + y - \frac{1}{x}$$

$$\frac{\partial f}{\partial y} = x + \frac{1}{y} - \frac{x}{y^2}$$

Las reglas de diferenciación que aprendimos anteriormente en este capítulo todavía se aplican. Observe que la  $d$  ha cambiado a  $\partial$ . Esto indica que la función,  $f$ , es de más de una variable. Además, observe que al calcular las respectivas derivadas, mantuvimos fija la otra variable como si fuera un parámetro. En cuanto al cálculo de derivadas parciales, eso es todo. Veamos algunos ejemplos más para ayudarte a concretar la idea.

Si  $f(x,y,z) = x^2 + y^{2+2+z} + 3xyz$ , podemos encontrar tres derivadas parciales:

$$\frac{\partial f}{\partial x} = 2x + 3yz$$

$$\frac{\partial f}{\partial y} = 2y + 3xz$$

$$\frac{\partial f}{\partial z} = 2z + 3xy$$

Las otras dos variables se consideran constantes. Esta es la razón por la que, por ejemplo, en la derivada parcial con respecto a  $x$ ,  $y$  se convierte en 0 y  $3xyz$  se convierte en  $3yz$ .

Si  $f(x,y,z, t) = \frac{xy}{zt} + y\sqrt{z} + \sqrt{xt}$ , tenemos cuatro derivadas parciales:

$$\frac{\partial f}{\partial x} = \frac{y}{zt} + 0 + \frac{1}{2}\sqrt{tx}^{-\frac{1}{2}}$$

$$= \frac{y}{zt} + \frac{1}{2}\sqrt{tx}^{-\frac{1}{2}}$$

$$= \frac{y}{zt} + \frac{1}{2}\frac{1}{\sqrt{t}}\frac{1}{x}$$

$$\frac{\partial f}{\partial y} = \frac{x}{zt} + \sqrt{z}$$

$$\frac{\partial f}{\partial z} = \frac{-xy}{tz^2} + \frac{1}{2}\frac{1}{\sqrt{z}}$$

$$\frac{\partial f}{\partial t} = \frac{-xy}{zt^2} + \frac{1}{2}\frac{1}{\sqrt{xt}}$$

Como ejemplo más complejo, considere  $f(x,y) = e^{xy} \cos x \sin y$ . el parcial  
Las derivadas se enumeran a continuación donde utilizamos la regla del producto en cada caso.

$$\frac{\partial f}{\partial x} = (ye \ xy \ \operatorname{sen} y)(\cos x) + e \ xy \ \operatorname{sen} y(-\operatorname{sen} x)$$

$$= e \ xy \ \operatorname{sen} y(y \cos x - \operatorname{sen} x)$$

$$\frac{\partial f}{\partial y} = (xe \ xy \ \cos x)(\sin y) + e \ xy \ \cos x \operatorname{acogedor} y$$

$$= e \ xy \ \cos x(x \ \operatorname{sen} y + \operatorname{cos} y)$$

### Derivados Parciales Mixtos

Al igual que con la derivada de una función de una sola variable, podemos tomar derivadas parciales de una derivada parcial. Estos se conocen como parciales mixtos. Además, tenemos más flexibilidad porque podemos cambiar con respecto a qué variable tomamos la siguiente derivada parcial. Por ejemplo, arriba vimos que la derivada parcial de  $f(x,y) = \frac{\partial^2 f}{\partial z^2} + y \sqrt{z} + \sqrt{x}t$  con respecto a  $z$  es

$$\frac{\partial F}{\partial z} = \frac{-xy \cdot y \frac{1}{2} t z^2}{\sqrt{z}} =$$

que sigue siendo una función de  $x, y, z$  y  $t$ . Por lo tanto, podemos calcular segundas derivadas parciales así:

$$\frac{\partial^2 F}{\partial z^2} = \frac{-y}{tz^2}$$

$$\frac{\partial^2 F}{\partial y \partial z} = \frac{-x}{tz^2} + \frac{1}{2\sqrt{z}} =$$

$$\frac{\partial^2 F}{\partial z \partial t} = (-xy \frac{1}{2}) (-1 \frac{1}{t^2}) =$$

$$= \frac{-xy}{t^2 z^2}$$

$$\frac{\partial^2 F}{\partial z^2} (-xy \frac{1}{t}) (-2) (\frac{1}{z^3}) + (y \frac{1}{2}) (-1 \frac{1}{2}) (\frac{1}{z^{3/2}})$$

$$= \frac{2xy}{tz^3} - \frac{1}{y 4z^{3/2}}$$

Explicaré la notación. Comenzamos con la derivada parcial de  $f$  con respecto a  $z$ , así que escribimos  $f/z$ . Luego, a partir de este punto de partida, vamos tomando otras derivadas parciales. Entonces, si queremos denotar el parcial con respecto a  $x$ , lo pensamos de esta manera:

$$\overline{X} \left( \frac{F}{z} \right) = \frac{\partial^2 F}{xz}$$

donde podemos pensar en el operador de derivada parcial “multiplicando” el “numerador” y el “denominador” como una fracción. Sin embargo, para ser claros, estas no son fracciones; la notación acaba de heredar el sabor de una fracción de sus orígenes de pendiente. Aún así, si la mnemónica es útil, entonces es útil. Para una segunda derivada parcial, la variable con respecto a la cual se toma está a la izquierda. Además, si las variables son las mismas, se utiliza un exponente (de algún tipo)<sup>2</sup> como en

La regla de la cadena para derivados parciales

Para aplicar la regla de la cadena a derivadas parciales, necesitamos rastrear todas las variables. Entonces, si tenemos  $f(x,y)$ , donde tanto  $x$  como  $y$  son funciones de otras variables,  $x(r,s)$  e  $y(r,s)$ , entonces podemos encontrar los parciales de  $f$  con respecto a  $r$  y  $s$  aplicando la regla de la cadena para cada variable,  $x$  e  $y$ . Específicamente,

$$\frac{\partial F}{\partial r} = \left( \frac{\partial F}{\partial s} \right) + \left( \frac{\partial f}{\partial x} \right) \left( \frac{\partial x}{\partial r} \right) + \left( \frac{\partial f}{\partial y} \right) \left( \frac{\partial y}{\partial r} \right)$$

$$\frac{\partial F}{\partial s} = \left( \frac{\partial F}{\partial r} \right) - \left( \frac{\partial f}{\partial x} \right) \left( \frac{\partial x}{\partial s} \right) - \left( \frac{\partial f}{\partial y} \right) \left( \frac{\partial y}{\partial s} \right)$$

Como ejemplo, sea  $f(x,y) = x^3 + y^3$  con  $x(r,s) = 3r+2s$  y  $y(r,s) = r^2 - 3s$ .

Ahora encuentre  $f/r$  y  $f/s$ . Para encontrar estos parciales, necesitaremos calcular seis expresiones,

$$\frac{\partial f}{\partial x} = 3x; \quad \frac{\partial f}{\partial y} = 3y; \quad \frac{\partial x}{\partial r} = 3; \quad \frac{\partial x}{\partial s} = 2; \quad \frac{\partial y}{\partial r} = 2r; \quad \frac{\partial y}{\partial s} = -3$$

para que los parciales deseados sean

$$\frac{F}{r} = \left( \frac{F}{(xr)} \right) \frac{1}{r} + \left( f \right) \frac{1}{(yx)} \frac{1}{r}$$

$$= (3x^2)(3) + (3 \text{ años}^2)(2r)$$

$$2 = 9x + 6 \text{ años}^2 r$$

$$= 9(3r + 2s)^2 + 6(r)^2 - 3s)^2 r$$

$$\frac{F}{s} = \left( \frac{F}{(xs)} \right) \frac{1}{s} + \left( f \right) \frac{1}{(yx)} \frac{1}{s}$$

$$= (3x^2)(2) + (3 \text{ años}^2)(-3)$$

$$2 = 6x - 9 \text{ años}^2$$

$$= 6(3r + 2s)^2 - 9(r)^2 - 3s)^2$$

Al igual que con una función de una sola variable, la regla de la cadena para funciones de más de una variable es recursiva, de modo que si  $r$  y  $s$  fueran funciones de otra variable, podríamos aplicar la regla de la cadena una vez más para encontrar el parcial de  $f$  con respecto a esa variable. Por ejemplo, si tenemos  $x(r, s)$ ,  $y(r, s)$ , con  $r(w)$ ,  $s(w)$ , encontramos  $f/w$  usando

$$\frac{F}{w} = \left( \frac{F}{(xr)} \right) \frac{1}{(rr)} \left( \frac{1}{(rw)} \right) + \left( \frac{f}{(yx)} \right) \frac{1}{(yx)} \left( \frac{1}{(xs)} \right) \frac{1}{(ss)}$$

$$\left( \frac{F}{(sw)} \right) \frac{1}{y} \frac{1}{(sy)}$$

Al final, debemos recordar que  $f/w$  nos dice cómo cambiará  $f$  para un pequeño cambio en  $w$ . Usaremos este hecho durante el descenso de gradiente.

Esta sección se ocupó del cálculo mecánico de derivadas parciales. Pasemos a explorar más sobre el significado detrás de estas cantidades. Esto nos llevará a la idea de un gradiente.

## Degrados

En el Capítulo 8, profundizaremos en la representación del cálculo matricial que utilizamos en el aprendizaje profundo. Sin embargo, antes de hacer eso, concluiremos este capítulo introduciendo la idea de gradiente. El gradiente se basa en las derivadas que hemos estado calculando. En resumen, el gradiente nos dice cómo cambia una función de más de una variable y en qué dirección cambia la mayoría.

### Calculando el gradiente Si

tenemos  $f(x,y,z)$ , vimos arriba cómo calcular las derivadas parciales con respecto a cada una de las variables. Si interpretamos las variables como posiciones en ejes de coordenadas, vemos que  $f$  es una función que devuelve un escalar, un único número, para cualquier posición en el espacio 3D,  $(x,y,z)$ . Incluso podríamos ir tan lejos como para escribir  $f(\mathbf{x})$  donde  $\mathbf{x} = (x,y,z)$  para reconocer que  $f$  es una función de una entrada vectorial.

Como en capítulos anteriores, usaremos letras minúsculas y negrita para representar vectores,  $\mathbf{x}$ . Tenga en cuenta que algunas personas usan  $\mathbf{x}$  para representar vectores.

Podemos escribir vectores horizontalmente, como un vector fila, como en el anterior párrafo, o verticalmente,

$$\begin{matrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{matrix}$$

como vector de columna, donde también hemos usado corchetes en lugar de paréntesis. Cualquiera de las notaciones es aceptable. A menos que seamos intencionalmente descuidados, generalmente cuando hablamos de un vector en el código, asumiremos que nuestros vectores son vectores de columna. Esto significa que un vector es una matriz con  $n$  filas y una columna,  $n \times 1$ .

Una función que acepta una entrada vectorial y devuelve un único número como salida se conoce como campo escalar. El ejemplo canónico de un campo escalar es la temperatura. Podemos medir la temperatura en cualquier punto de una habitación. Representamos la ubicación como un vector 3D relativo a algún punto de origen elegido, y la temperatura es el valor en ese punto, la energía cinética promedio de las moléculas en esa región. También podemos hablar de funciones que aceptan vectores como entrada y devuelven un vector como salida. Estos se conocen como campos vectoriales. En ambos casos, la parte de campo se refiere al hecho de que, sobre algún dominio adecuado, la función tiene un valor para todas las entradas.

El gradiente es la derivada de una función que acepta un vector como entrada. Matemáticamente, representamos el gradiente como una generalización de la idea de derivadas parciales en  $n$  dimensiones. Por ejemplo, en el espacio 3D, podemos escribir

—  
X

$$f(x) =$$

F  
Z

donde el operador de gradiente,  $\nabla$ , toma la derivada parcial de  $f$  a lo largo de cada una de sus dimensiones. El operador  $\nabla$  tiene varios nombres, como del, grad o nabla. Usaremos  $\nabla$  y lo llamaremos del cuando no estemos simplemente diciendo "operador de gradiente".

En general podemos escribir

$$y = f(x) = f(x_0, x_1, \dots, x_n) \equiv \frac{F}{x_0} \quad (7.7)$$

Analicemos la ecuación 7.7. Primero, tenemos una función,  $f$ , que acepta una entrada vectorial,  $x$ , y devuelve un valor escalar. A esta función le aplicamos el operador gradiente:

—  
x0

—  
x1

x2

xv

Este devuelve un vector ( $y$ ). El operador de gradiente convierte la salida escalar de  $f$  en un vector. Dediquemos un tiempo a pensar en lo que esto significa, en lo que nos dice sobre el valor del campo escalar en una posición determinada en el espacio. (Usaremos el espacio cuando trabajemos con vectores, incluso si no hay una forma significativa de visualizar el espacio. Una analogía con el espacio 3D es útil, pero solo llega hasta cierto punto; matemáticamente, la idea de espacio es más general).

Como ejemplo, considere una función en el espacio 2D,  $f(x) = f(x,y) = x^2 + y^2$ . El gradiente de  $f$  es entonces

$$\begin{aligned} \frac{\partial f}{\partial x} &= \frac{\partial}{\partial x} (x^2 + xy + y^2) \\ \frac{\partial f}{\partial y} &= \frac{\partial}{\partial y} (x^2 + xy + y^2) = [2x+xy]2y \end{aligned} \quad (7.8)$$

Dado que  $f$  es un campo escalar, cada punto en el plano 2D tiene un valor de función. Esta es la salida de  $f(x) = f(x,y)$ . Entonces, podemos trazar  $f$  en 3D para mostrarnos una superficie que cambia con la posición. El gradiente, sin embargo, nos da un conjunto de ecuaciones. Estas ecuaciones en conjunto nos dicen la dirección y magnitud del cambio en el valor de la función en un punto,  $x = (x,y)$ .

Para una función de una sola variable, solo hay una pendiente en cada punto.

Mire nuevamente la línea tangente de la Figura 7-1. En el punto  $x$  sólo hay una pendiente. El signo de la derivada en  $x$  da la dirección de la pendiente y el valor absoluto de la derivada da la magnitud (inclinación) de la pendiente.

Sin embargo, una vez que cambiamos a más de una dimensión, nos encontramos con un enigma. En lugar de una sola pendiente tangente a la función, ahora tenemos un número infinito. Podemos imaginar una recta tangente a la función en algún punto y que la recta apunte en cualquier dirección que deseemos. La pendiente de la línea nos dice cómo cambia el valor de la función en esa dirección particular. Podemos encontrar el valor de este cambio a partir de la derivada direccional, el producto escalar entre el gradiente en el punto considerado y un vector unitario en la dirección que nos interesa:

$$\text{D}_u f(x) \equiv u \cdot f(x) = u^T f(x) = u \cdot f(x) \cos$$

donde  $u$  es un vector unitario en una dirección particular,  $f(x)$  es el gradiente de la función en el punto  $x$  y  $\theta$  es el ángulo entre ellos. La derivada direccional se maximiza cuando se maximiza  $\cos$ , y esto sucede en  $\theta = 0$ .

Por lo tanto, la dirección del cambio máximo en una función en cualquier punto es el gradiente en ese punto.

Como ejemplo, elegimos un punto en el espacio 2D, digamos  $x = (x,y) = (0.5, -0.4)$  con  $f(x,y) = x^2 + y^2$ . Entonces, el valor de la función es  $(0.5)^2 + (-0.4)^2 = 0.25 + 0.16 = 0.41$ , un escalar. Sin embargo, el gradiente en este punto es

$$f(0.5, -0.4) = [2x + 2y]_{(0.5, -0.4)} = [2(0.5) + 2(-0.4)] = [1 - 0.8] = 0.2$$

Por lo tanto, ahora sabemos que en el punto  $(0.5, -0.4)$ , la dirección del mayor cambio es  $f$  en la dirección  $(0.6, -0.3)$  y tiene una magnitud de  $\sqrt{(0.6)^2 + (-0.3)^2} \approx 0.67$ .

## Visualizando el degradado

Hagamos todo esto menos abstracto. La parte superior de la Figura 7-4 muestra una gráfica de  $f(x,y) = x^2 + y^2$ .

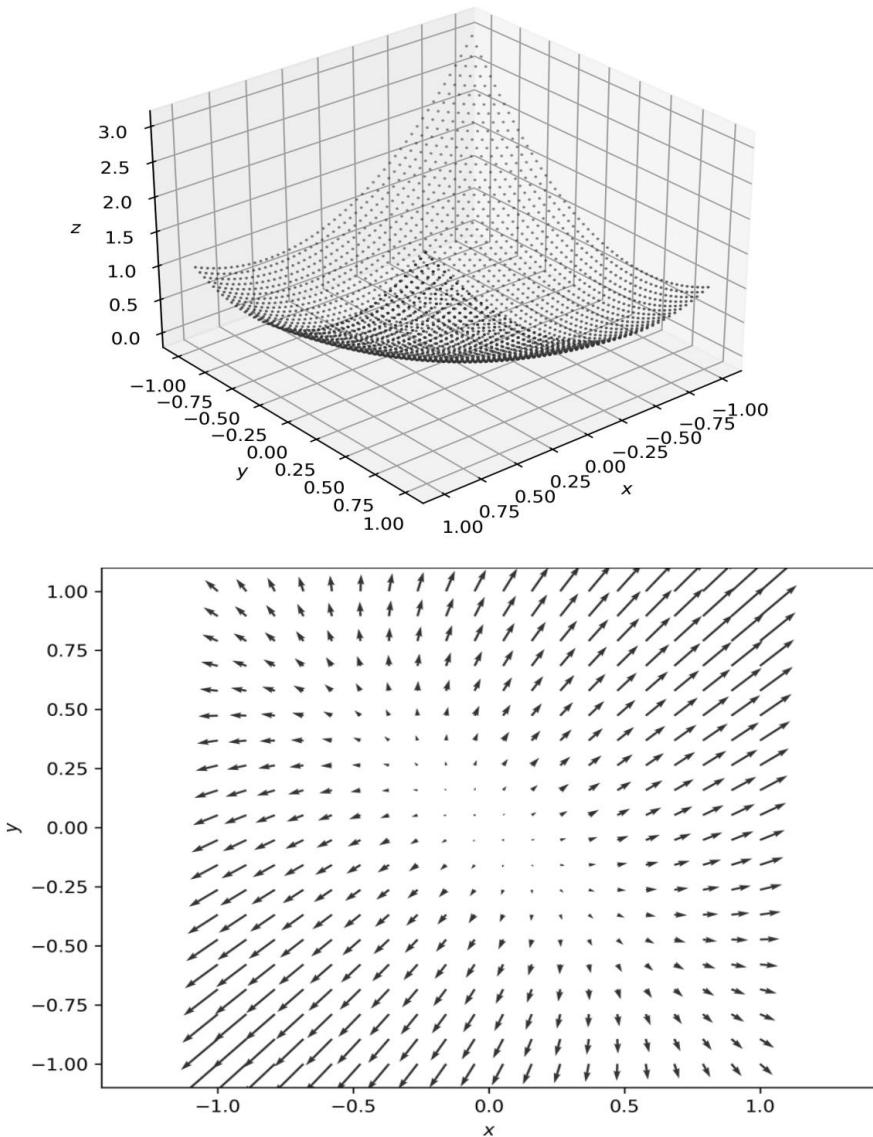


Figura 7-4: Gráfico del campo  $x^2 + xy + y^2$  (arriba) y una proyección 2D del gradiente asociado (abajo)

El código para generar este gráfico es sencillo:

---

```
importar numpy
como np x = np.linspace (-1.0,1.0,50)
```

```

y = np.linspace(-1.0,1.0,50) xx
= []; aa = []; zz = [] para i en
el rango(50): para j
    en el rango(50):
        xx.append(x[i])
        yy.append(y[j])
        zz.append(x[i]*x[i ]+x[i]*y[j]+y[j]*y[j]) x =
np.array(xx) y =
np.array(yy) z =
np.array(zz)

```

---

Aquí, hacemos un bucle explícito para generar el conjunto de puntos del diagrama de dispersión, x, y, z, para mostrar claramente lo que está sucediendo. Primero, usamos NumPy para generar vectores de 50 puntos espaciados uniformemente [-1, 1] en x. Luego configuramos un bucle doble para que cada x se empareje con cada y para calcular el valor de la función, z. Las listas temporales xx, yy y zz contienen los tripletes. Finalmente, convertimos las listas a matrices NumPy para trazar.

El código para generar el diagrama de dispersión es

```

de mpl_toolkits.mplot3d importar Axes3D
importar matplotlib.pyplot como plt
fig = plt.figure() ax =
fig.add_subplot(111, proyección='3d')
ax.scatter(x, y, z, marcador='.', s =2, color='b')
ax.view_init(30,20)
plt.draw()
plt.show()

```

---

Primero cargamos la extensión matplotlib para el trazado 3D y luego configuramos la trama secundaria para una proyección 3D. El gráfico en sí está hecho con ax.scatter, mientras que ax.view\_init y plt.draw rotan el gráfico para darnos una mejor vista de la forma de la función antes de mostrarla.

En la parte inferior de la Figura 7-4, vemos un gráfico vectorial del campo de gradiente para x. Este gráfico muestra la dirección y magnitud relativa de el vector gradiente en una cuadrícula de puntos, (x,y). Recuerde, el gradiente es un campo vectorial, por lo que cada punto en el plano xy tiene un vector asociado que apunta en la dirección del mayor cambio en el valor de la función. Mentalmente, podemos ver cómo se relaciona el gráfico vectorial con el gráfico de función en la parte superior de la Figura 7-4, donde los valores de función cercanos a (-1, -1) y (1, 1) cambian rápidamente, mientras que para los puntos cercanos a (0, 0) están cambiando lentamente.

El código para generar el gráfico de campo vectorial es

```

fig = plt.figure() ax =
fig.add_subplot(111) x =
np.linspace(-1.0,1.0,20) y =
np.linspace(-1.0,1.0,20) xv, yv
= np.meshgrid(x , y, indexing='ij', sparse=False) dx = 2*xv + yv
dy = 2*yv + xv

```

---

---

```
ax.quiver(xv, yv, dx, dy, color='b')
plt.axis('equal')
plt.show()
```

---

Primero definimos la figura (fig) y la subrama para 2D (sin palabra clave de proyección). Entonces, necesitamos una cuadrícula de puntos. Arriba, hicimos un bucle para obtener esta cuadrícula para poder comprender qué era necesario generar. Aquí, usamos NumPy para generar la cuadrícula a través de np.meshgrid. Tenga en cuenta que pasamos a np.meshgrid los mismos vectores xey que teníamos arriba para definir el dominio.

Las dos líneas siguientes son una implementación directa del gradiente de f, Ecuación 7.8. Estos son los vectores que queremos trazar, donde dx y dy nos dan la dirección y la magnitud, mientras que xv e yv son el conjunto de puntos de entrada: 400 en total.

La trama utiliza ax.quiver (ya que traza flechas). Los argumentos son la cuadrícula de puntos (xv, yv) y los valores xey asociados de los vectores en esos puntos (dx, dy). Finalmente, nos aseguramos de que los ejes sean iguales (plt.axis) para evitar deformar la visualización del vector y luego mostramos el gráfico.

Concluiremos aquí nuestra introducción a los gradientes. Los veremos nuevamente a lo largo del resto del libro, en la notación del Capítulo 8 y en las discusiones sobre el descenso de gradiente del Capítulo 11.

## Resumen

Este capítulo introdujo los conceptos principales del cálculo diferencial. Comenzamos con la noción de pendiente y aprendimos la diferencia entre rectas secantes y tangentes para una función de una sola variable. Luego definimos formalmente la derivada como la pendiente de una recta secante cuando se acerca a un solo punto.

A partir de ahí aprendimos las reglas básicas de diferenciación y vimos cómo aplicarlas.

A continuación, aprendimos sobre los mínimos y máximos de una función y cómo encontrar estos puntos usando derivadas. Luego introdujimos las derivadas parciales como una forma de calcular derivadas para funciones de más de una variable. Luego, las derivadas parciales nos llevaron al gradiente, que convierte un campo escalar en un campo vectorial y nos indica la dirección en la que la función cambia más. Calculamos un gradiente de ejemplo en 2D y vimos cómo generar gráficos que muestren la relación entre la función y el gradiente. Aprendimos el hecho crucial de que el gradiente de una función apunta en la dirección del cambio máximo en el valor de la función en un punto.

Continuemos nuestra exploración de las matemáticas detrás del aprendizaje profundo y avancemos al mundo del cálculo matricial.



# 8

## CÁLCULO MATRICIAL



El capítulo 7 nos presentó el cálculo diferencial. En este capítulo, discutiremos la matriz. cálculo, que extiende la diferenciación a funciones que involucran vectores y matrices.

El aprendizaje profundo trabaja ampliamente con vectores y matrices, por lo que hace sentido desarrollar una notación y un enfoque para representar derivadas que involucran estos objetos. Eso es lo que nos proporciona el cálculo matricial. Vimos un indicio de esto. al final del Capítulo 7, cuando introdujimos el gradiente para representar el derivada de una función escalar de un vector: una función que acepta un vector argumento y devuelve un escalar,  $f(x)$ .

Comenzaremos con la tabla de derivadas del cálculo matricial y sus definiciones. A continuación, examinaremos algunas identidades que involucran derivadas matriciales. Los matemáticos aman las identidades; Sin embargo, para preservar nuestra cordura, sólo haremos considerar un puñado. Algunas matrices especiales surgen del cálculo matricial, a saber, el jacobiano y el hessiano. Te toparás con estas dos matrices durante tu estancia en el aprendizaje profundo, por lo que las consideraremos a continuación en el siguiente capítulo. contexto de optimización. Recuerde que entrenar una red neuronal es, fundamentalmente, un problema de optimización, por lo que comprender qué son estas matrices especiales representamos y cómo los utilizamos es especialmente importante. Cerraremos el capítulo con algunos ejemplos de derivadas matriciales.

## Las Fórmulas

La tabla 8-1 resume las derivadas del cálculo matricial que exploraremos en este capítulo. Estos son los que se utilizan habitualmente en la práctica.

Tabla 8-1: Derivadas del cálculo matricial

Matriz vectorial escalar	
Escalar $f / x$	$f/x$
Vector $f / x$	$f/x$
Matriz $f / X$	$f/X$

Las columnas de la Tabla 8-1 representan la función, es decir, el retorno valor. Observe que utilizamos tres versiones de la letra f: regular, negrita y mayúscula. Usamos f si el valor de retorno es un escalar, f si es un vector y F si es una matriz. El

Las filas de la Tabla 8-1 son las variables con respecto a las cuales se calculan las derivadas.

a. Se aplica la misma notación: x es un escalar, x es un vector y X es una matriz.

La tabla 8-1 define seis derivadas, pero hay nueve celdas en la tabla.

Si bien es posible definirlos, los derivados restantes no están estandarizados ni se usan con suficiente frecuencia como para que valga la pena cubrirlos. Eso es bueno para nosotros ya que los seis son un desafío suficiente para nuestro cerebro matemático.

La primera derivada de la tabla 8-1, la que está en la parte superior izquierda, es la derivada normal del capítulo 7, una función que produce un escalar con respecto a un escalar. (Consulte el Capítulo 7 para obtener todo lo que necesita saber sobre el estándar. diferenciación.)

Cubriremos los cinco derivados restantes en las secciones siguientes. Definimos cada uno en términos de derivadas escalares. Primero mostraremos la definición.

y luego explica qué representa la notación. La definición te ayudará.

construye un modelo en tu cabeza de cuál es la derivada. Sospecho que por el

Al final de esta sección estarás prediciendo las definiciones de antemano.

Sin embargo, antes de comenzar, hay una complicación que debemos analizar.

El cálculo matricial requiere mucha notación, pero no existe un acuerdo universal sobre la notación. Hemos visto esto antes con las muchas formas de indicar diferenciación. Para el cálculo matricial, los dos enfoques son el diseño del numerador o el diseño del denominador. Aunque disciplinas específicas parecen favorecer unas sobre otras.

las excepciones son casi la norma, al igual que mezclar notaciones. Para el aprendizaje profundo,

Una lectura no científica por mi parte parece indicar una ligera preferencia por

diseño del numerador, así que eso es lo que usaremos aquí. Sólo ten en cuenta que hay

dos formas por ahí. Normalmente uno es la transposición del otro.

Una función vectorial mediante un argumento escalar

Una función vectorial que acepta un argumento escalar es nuestra primera derivada; ver Tabla 8-1, segunda columna de la primera fila. Escribimos una función como  $f(x)$

para indicar un argumento escalar,  $x$ , y una salida vectorial,  $f$ . Funciones como  $f$  toman un escalar y lo asignan a un vector multidimensional:

$$f \rightarrow m$$

Aquí,  $m$  es el número de elementos en el vector de salida. Funciones como  $f$  se conocen como funciones vectoriales con argumentos escalares.

Una curva paramétrica en el espacio 3D es un excelente ejemplo de tal función. Esas funciones a menudo se escriben como

$$f(x) = f_0(x) \quad x \quad f_1(x) \quad y \quad f_2(x) \quad z$$

donde  $x$ ,  $y$  y  $z$  son vectores unitarios en las direcciones  $x$ ,  $y$  y  $z$ .

La Figura 8-1 muestra un trazado de una curva paramétrica 3D,

$$f(t) = t \cos(t) \quad x + t \sin(t) \quad y + t \quad z \quad (8.1)$$

donde, a medida que  $t$  varía, los valores de los tres ejes también varían para trazar la espiral. Aquí, cada valor de  $t$  especifica un único punto en el espacio 3D.

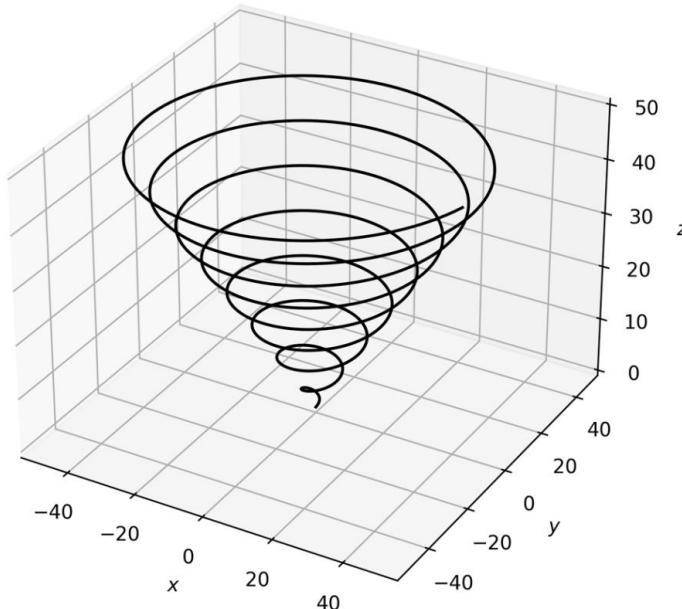


Figura 8-1: Una curva paramétrica 3D

En notación de cálculo matricial, no escribimos  $f$  como se muestra en la Ecuación 8.1. En cambio, escribimos  $f$  como un vector columna de las funciones,

$$f(t) = \begin{pmatrix} t \cos(t) \\ t \sin(t) \\ t \end{pmatrix}$$

y en general,

$$f(x) = \begin{matrix} f_0(x) \\ f_1(x) \\ \vdots \\ f_{n-1}(x) \end{matrix}$$

para  $f$  con  $n$  elementos.

La derivada de  $f(x)$  se conoce como vector tangente. ¿Cómo se ve la derivada? Dado que  $f$  es un vector, podríamos esperar que la derivada de  $f$  sean las derivadas de las funciones que representan cada elemento de  $f$ , y estaríamos en lo cierto:

$$\frac{F}{x} = \begin{matrix} f_0/x \\ f_1/x \\ \vdots \\ f_{n-1}/x \end{matrix}$$

Veamos un ejemplo sencillo. Primero, definiremos  $f(x)$ , y luego la derivado:

$$f(x) = [2x^2 - 3x - 3], \quad \frac{F}{x} = [4x - 3]$$

Aquí, cada elemento de  $F/x$  es la derivada de la función correspondiente en  $f$ .

Una función escalar mediante un argumento vectorial

En el Capítulo 7, aprendimos que una función que acepta una entrada vectorial pero devuelve un escalar es un campo escalar:

$$f \text{ metro} \rightarrow$$

También aprendimos que la derivada de esta función es el gradiente. En notación de cálculo matricial, escribimos  $F/x$  para  $f(x)$  como

$$\frac{F}{x} = [f_x^0 \quad f_x^1 \quad \dots \quad f_x^{m-1}]$$

donde  $x = [x_0 \ x_1 \ \dots \ x_{m-1}]$  esas variables y  $f$  es una función de variables.

Observe que, debido a que decidimos utilizar el enfoque de diseño del numerador,  $f/x$  se escribe como un vector de fila. Entonces, para ser fiel a nuestra notación, tenemos que escribir

$$f(x) = [ f_x ]^T$$

convirtiendo el vector de fila en un vector de columna para que coincida con el gradiente. Recuerde que  $\nabla$  es el símbolo del gradiente; Vimos un ejemplo del gradiente en el Capítulo 7.

Una función vectorial por un vector Si

la derivada de una función vectorial por un escalar produce un vector columna, y la derivada de una función escalar por un vector da como resultado un vector fila, ¿la derivada de una función vectorial por un vector produce una matriz? La respuesta es sí. En este caso, estamos contemplando  $f/x$  para  $f(x)$ , una función que acepta una entrada vectorial y devuelve un vector.

La convención de diseño del numerador nos dio un vector de columna para la derivación. tiva de  $f(x)$ , lo que implica que necesitamos una fila para cada una de las funciones en  $f$ . De manera similar, la derivada de  $f(x)$  produjo un vector fila. Por lo tanto, fusionar los dos nos da la derivada de  $f(x)$ :

$$\frac{F}{x} = \begin{matrix} \frac{f_0}{x_0} & \frac{f_0}{x_1} & \frac{f_0}{x_{m-1}} \\ \frac{f_1}{x_0} & \frac{f_1}{x_1} & \frac{f_1}{x_{m-1}} \\ \vdots & \vdots & \vdots \\ \frac{f_{n-1}}{x_0} & \frac{f_{n-1}}{x_1} & \frac{f_{n-1}}{x_{m-1}} \end{matrix} \quad (8.2)$$

Esto es para una función,  $f$ , que devuelve un vector de  $n$  elementos y acepta un vector de  $m$  elementos,  $x$ , como argumento:

$$f : m \rightarrow n$$

Cada fila de  $f$  es una función escalar de  $x$ , por ejemplo,  $f_0(x)$ . por lo tanto, nosotros Puede escribir la ecuación 8.2 como

$$\frac{F}{x} = \begin{matrix} f_0(x) \\ f_1(x) \\ \vdots \\ f_{n-1}(x) \end{matrix} \quad (8.3)$$

Esto nos da la matriz como una colección de gradientes, uno para cada función escalar en  $f$ . Volveremos a esta matriz más adelante en este capítulo.

Una función matricial por un escalar

Si  $f(x)$  es una función que acepta un argumento escalar pero devuelve un vector, entonces estaríamos en lo correcto al suponer que  $F(x)$  puede considerarse como una función que acepta un argumento escalar pero devuelve una matriz:

$$F \rightarrow n \times m$$

Por ejemplo, supongamos que  $F$  es una matriz  $n \times m$  de funciones escalares:

$$F = \begin{matrix} f_{00}(x) & f_{01}(x) & f_{0,m-1}(x) & f_{11}(x) \\ f_{10}(x) & f_{1,m-1}(x) \\ & f_{n-1,0}(x) & f_{n-1,1}(x) & f_{n-1,m-1}(x) \end{matrix}$$

La derivada con respecto al argumento,  $x$ , es sencilla:

$$\frac{F}{x} = \begin{matrix} \frac{f_{00}}{x} & \frac{f_{01}}{x} & \frac{f_{0,m-1}}{x} \\ \frac{f_{10}}{x} & \frac{f_{11}}{x} & \frac{f_{1,m-1}}{x} \\ \frac{f_{n-1,0}}{x} & \frac{f_{n-1,1}}{x} & \frac{f_{n-1,m-1}}{x} \end{matrix}$$

Como vimos anteriormente, la derivada de una función vectorial por un escalar se llama vector tangente. Entonces, por analogía, la derivada de una función matricial por un escalar es la matriz tangente.

Una función escalar mediante una matriz

Ahora consideremos  $f(X)$ , una función que acepta una matriz y devuelve un escalar:

$$f \rightarrow n \times m$$

Estaríamos en lo cierto al pensar que la derivada de  $f$  con respecto a la matriz,  $X$ , es en sí misma una matriz. Sin embargo, para ser fiel a nuestro diseño del numerador

Por convención, la matriz resultante no está dispuesta como  $X$ , sino como  $X$  la transpuesta , de  $X$ .

¿ Por qué utilizar la transpuesta de  $X$  en lugar de la propia  $X$  ? Para responder a la pregunta, Necesitamos recordar cómo definimos  $f/x$ . Allí, aunque  $x$  es un vector columna, según la convención estándar, dijimos que la derivada es un vector fila . Usamos  $x$  como orden. Por lo tanto, para ser coherentes, necesitamos ordenar  $f/X$  mediante la transposición de  $X$  y cambiar las columnas de  $X$  en filas en la derivada. Como resultado, tenemos la siguiente definición.

$$\frac{\underline{\underline{F}}}{X} = \begin{matrix} \overline{\overline{fx_{00}}} & \overline{\overline{x_{10}}} & \overline{\overline{fx_{n-1,0}}} \\ \overline{\overline{fx_{01}}} & \overline{\overline{fx_{11}}} & \overline{\overline{f_{xn-1,1}}} \\ \overline{\overline{fx_{0,m-1}}} & \overline{\overline{fx_{1,m-1}}} & \overline{\overline{fx_{n-1,m-1}}} \end{matrix} \quad (8.4)$$

Ésta es una matriz de salida de  $m \times n$  para la matriz de entrada de  $n \times m$ , X.

La ecuación 8.4 define la matriz de gradiente, que, para las matrices, desempeña un papel similar al del gradiente,  $\nabla f(x)$ . La ecuación 8.4 también completa nuestra recopilación.ción de derivadas del cálculo matricial. Pasemos a considerar algunas identidades derivadas de matrices.

## Las identidades

El cálculo matricial involucra escalares, vectores, matrices y funciones de los mismos, que a su vez devuelven escalares, vectores o matrices, lo que implica que existen muchas identidades y relaciones. Sin embargo, aquí nos concentraremos en identidades básicas que muestran la relación entre el cálculo matricial y el cálculo diferencial del Capítulo 7.

Cada una de las siguientes subsecciones presenta identidades relacionadas con el tipo específico de derivado indicado. Las identidades cubren relaciones fundamentales y, cuando corresponde, la regla de la cadena. En todos los casos, los resultados siguen el esquema de disposición del numerador que hemos utilizado a lo largo del capítulo.

Una función escalar mediante un

vector Comenzamos con identidades relacionadas con una función escalar con una entrada vectorial. Si no se especifica lo contrario, f y g son funciones de un vector, x, y devuelven un escalar. Un vector constante que no depende de x se da como a, y a denota una constante escalar.

Las reglas básicas son intuitivas:

$$\overline{\overline{x}}(af) = a \overline{\overline{\frac{f}{x}}} \quad (8.5)$$

y

$$\overline{\overline{x}}(f+g) = \overline{\overline{\frac{f}{x}}} + \overline{\overline{\frac{g}{x}}} \quad (8.6)$$

Estos muestran que la multiplicación por una constante escalar actúa como lo hizo en el capítulo 7, al igual que la linealidad de la derivada parcial.

La regla del producto también funciona como esperamos:

$$\overline{\underline{x}} \overline{\underline{f}g} (\overline{\underline{fg}}) = \overline{\underline{f}} + \overline{\underline{g}} \overline{\underline{a}} \overline{\underline{x}} \quad (8.7)$$

Hagamos una pausa aquí y recordemos las entradas y salidas de las ecuaciones anteriores. Sabemos que la derivada de una función escalar mediante un argumento vectorial es un vector fila en nuestra notación. Entonces, la ecuación 8.5 devuelve un vector fila multiplicado por un escalar: cada elemento de la derivada se multiplica por a.

Como la diferenciación es un operador lineal, se distribuye sobre la suma, por lo que la ecuación 8.6 entrega dos términos, cada uno de los cuales es un vector fila generado por la derivada respectiva.

Para la ecuación 8.7, la regla del producto, el resultado nuevamente incluye dos términos. En cada caso, la derivada devuelve un vector fila, que se multiplica por un valor de función escalar, ya sea  $f(x)$  o  $g(x)$ . Por lo tanto, el resultado de la ecuación 8.7 también es un vector fila.

La regla de la cadena escalar por vector se convierte en

$$\overline{\underline{x}} \overline{\underline{f(g)}} = \overline{\underline{F}} \overline{\underline{\frac{g}{x}}} \quad (8.8)$$

donde  $f(g)$  devuelve un escalar y acepta un argumento escalar, mientras que  $g(x)$  devuelve un escalar y acepta un argumento vectorial. El resultado final es un vector de fila. Trabajemos con un ejemplo completo para demostrarlo.

Tenemos un vector,  $x = [x_0, x_1, x_2]$ ; una función de ese vector escrita en . De acuerdo a forma componente,  $g(x) = x_0 + x_1 x_2$ ; y una función de g,  $f(g) = g$  según la ecuación 8.8, la derivada de f con respecto a x es

$$\begin{aligned} \frac{\underline{F}}{\underline{x}} &= \frac{\underline{F}}{\underline{x}} \frac{\underline{\text{grado}}}{\underline{x}} \\ &= \frac{\underline{F}}{\underline{\text{grado}}} \frac{\underline{\text{grado}}}{\underline{[g(x)]}} \frac{\underline{\text{grado}}}{\underline{x_1}} \frac{\underline{\text{grado}}}{\underline{x_2}} \\ &= (2g) [ 1x2x1 ] \end{aligned}$$

$$= [2g \ 2gx2 \ 2gx1]$$

$$= [2g \ 2gx2 \ 2gx1]$$

$$= [2(x_0 + x_1 x_2) \ 2(x_0 + x_1 x_2)x_2 \ 2(x_0 + x_1 x_2)x_1]$$

$$= [2x_0 + 2x_1 x_2 \ 2x_0 x_2 + 2x_1 x_2 \ 2x_0 x_1 + 2x_1 x_2]$$

Para verificar nuestro resultado, podemos trabajar desde  $g(x) = x_0 + x_1 x_2$  y  $f(g) = g^2$  para encontrar  $f(x)$  directamente mediante sustitución. Hacer esto nos da

$$f(x) = x_0^2 + 2x_0 x_1 x_2 + x_1^2 x_2^2$$

de donde obtenemos

$$\begin{aligned} \frac{F}{X} &= [f \quad x_0 \quad \frac{F}{x_1} \quad \frac{F}{x_2}] \\ &= [2x_0 + 2x_1 x_2 \quad 2x_0 x_2 + 2x_1 x_2 \quad 2x_1 x_2] \end{aligned}$$

que coincide con el resultado que encontramos usando la regla de la cadena. Por supuesto, en este ejemplo simple, fue más fácil realizar la sustitución antes de tomar la derivada, pero de todos modos demostramos nuestro caso.

Sin embargo, no hemos terminado del todo con las identidades escalar por vector. El producto escalar toma dos vectores y produce un escalar, por lo que encaja con la forma funcional con la que estamos trabajando, aunque los argumentos del producto escalar sean vectores.

Por ejemplo, considere este resultado:

$$\frac{\partial}{\partial x} (a \cdot x) = \frac{\partial}{\partial x} (un \cdot x) = un \quad (8.9)$$

Aquí tenemos la derivada del producto escalar entre  $x$  y un vector  $a$  que no depende de  $x$ .

Podemos ampliar la ecuación 8.9, reemplazando  $x$  con una función vectorial,  $f(x)$ :

$$\frac{\partial}{\partial x} (a \cdot f) = \frac{\partial}{\partial x} (un \cdot f) = un \frac{\partial}{\partial x} \quad (8.10)$$

¿Cuál es la forma de este resultado? Supongamos que  $f$  acepta una entrada de elemento  $m$  y devuelve una salida vectorial de  $n$  elementos. Asimismo, supongamos que  $a$  es un vector de  $n$  elementos. De la ecuación 8.2 sabemos que la derivada  $f/x$  es una matriz de  $n \times m$ . Por lo tanto, el resultado final es un vector fila  $(1 \times n) \times (n \times m) \rightarrow 1 \times m$ .

¡Bien! Sabemos que la derivada de una función escalar por un vector debe ser un vector de fila cuando se usa la convención de diseño del numerador.

Finalmente, la derivada del producto escalar de dos funciones con valores vectoriales,  $f$  y  $g$ , es

$$\frac{\partial}{\partial x} (f \cdot g) = \frac{\partial}{\partial x} (F \text{ gramo}) = f \frac{\text{gramo}}{X^+ \text{ gramo}} \frac{F}{X} \quad (8.11)$$

Si la ecuación 8.10 es un vector fila, entonces la suma de dos términos como este también es un vector fila.

Una función vectorial mediante un escalar La diferenciación de vector por escalar, Tabla 8-1, primera fila, segunda columna, es menos común en el aprendizaje automático, por lo que solo examinaremos algunas identidades. Las primeras son multiplicaciones por constantes:

$$\frac{\partial}{\partial x} (af) = a \frac{\partial f}{\partial x}$$

$$\frac{\partial}{\partial x} (Af) = A \frac{\partial f}{\partial x}$$

Nota, podemos multiplicar a la izquierda por una matriz, ya que la derivada es una columna vector.

La regla de la suma todavía se aplica,

$$\frac{\partial}{\partial x} (f + g) = \frac{\partial F}{\partial x} + \frac{\partial g}{\partial x}$$

al igual que la regla de la cadena,

$$\frac{\partial}{\partial x} (f(g)) = \frac{\partial F}{\partial x} \Big|_{g(x)} \quad (8.12)$$

La ecuación 8.12 es correcta, ya que la derivada de un vector por un escalar es un vector columna y la derivada de un vector por un vector es una matriz. Por lo tanto, multiplicar la matriz de la derecha por un vector columna devuelve un vector columna, como se esperaba.

Vale la pena conocer otras dos derivadas que involucran productos escalares con respecto a un escalar. La primera es similar a la ecuación 8.11 pero con dos funciones de un escalar con valores vectoriales:

$$\frac{\partial}{\partial x} (f \cdot g) = f \cdot \frac{\partial g}{\partial x} + \frac{\partial f}{\partial x} \cdot g$$

$$\text{gramo} = f \cdot \frac{\partial g}{\partial x} + \frac{\partial f}{\partial x} \cdot g$$

La segunda derivada se refiere a la composición de  $f(g)$  y  $g(x)$  con respecto a  $x$ :

$$\frac{\partial}{\partial x} (f(g)) = \frac{\partial F}{\partial x} \Big|_{g(x)} \cdot \frac{\partial g}{\partial x} = \frac{\partial F}{\partial x} \Big|_{g(x)} \frac{\partial g}{\partial x}$$

que es el producto escalar de un vector fila y un vector columna.

Una función vectorial mediante un vector

Las derivadas de funciones con valores vectoriales y argumentos vectoriales son comunes en física e ingeniería. En el aprendizaje automático, aparecen durante la retropropagación, por ejemplo, en la derivada de la función de pérdida. Comencemos con algunas identidades sencillas:

$$\overline{\overline{X}}(af) = a \quad \frac{f}{X}$$

$$\overline{\overline{X}}(Af) = A \quad \frac{F}{X}$$

y

$$\overline{\overline{X}}(f+g) = \frac{F}{X} + \frac{g}{X}$$

donde el resultado es la suma de dos matrices.

La regla de la cadena es la siguiente y funciona como lo hizo anteriormente para escalar por vector y derivadas vector por escalar:

$$\overline{\overline{X}}(fg) = \frac{F}{X} \frac{g}{X}$$

siendo el resultado el producto de dos matrices.

Una función escalar mediante una matriz

Para funciones de una matriz,  $X$ , que devuelven un escalar, tenemos la forma habitual para la regla de la suma:

$$\overline{\overline{X}}(f+g) = \frac{f}{X} + \frac{g}{X}$$

siendo el resultado la suma de dos matrices. Recuerde, si  $X$  es una matriz de  $n \times m$ , la derivada en la notación de disposición del numerador es una matriz de  $m \times n$ .

La regla del producto también funciona como se espera:

$$\overline{\overline{X}}(fg) = f \quad \overline{\overline{X}}g \quad \overline{\overline{X}}$$

Sin embargo, la regla de la cadena es diferente. Depende de  $f(g)$ , una función escalar que acepta una entrada escalar, y  $g(X)$ , una función escalar que acepta una entrada matricial. Con esta restricción, la forma de la regla de la cadena resulta familiar:

$$\overline{\overline{X}}(fg) = \frac{F}{X} \frac{g}{X} \quad (8.13)$$

Veamos la ecuación 8.13 en acción. Primero, necesitamos X, una matriz de  $2 \times 2$ :

$$X = [x_0 \ x_1 \\ x_2 \ x_3]$$

A continuación, necesitamos  $\frac{f}{X}$  y  $g(X) = x_0 x_3 + x_1 x_2$ . Observe que mientras  $g(X)$  que  $f(g) =$  acepte una entrada matricial, el resultado es un escalar calculado a partir de los valores de la matriz.

Para aplicar la regla de la cadena, necesitamos dos derivadas,

$$\frac{f}{g} = gg$$

$$\frac{g}{X} = [g/x_0 \ g/x_1 \\ g/x_2 \ g/x_3]$$

donde nuevamente usamos el diseño del numerador para el resultado.

Para encontrar el resultado general, calculamos

$$\frac{F}{X} = \frac{fg}{X}$$

$$= g [x_0 \ x_1 \\ x_2 \ x_3]$$

$$= [x_0 g \ x_1 g \\ x_2 g \ x_3 g]$$

$$= [x_0 x_3 + x_1 x_2 \ x_0 x_3 + x_1 x_2]$$

Para comprobarlo, combinamos las funciones para escribir una sola función,  $f(X) = (x_0 x_3 + x_1 x_2)^2$ , y calcular la derivada usando la regla de la cadena estándar para cada elemento de la matriz resultante. esto nos da

$$\frac{F}{X} = [f/x_0 \ f/x_1 \ f/x_2 \ f/x_3 \\ f/x_1 \ f/x_0 \ f/x_3 \ f/x_2 \ f/x_2 \ f/x_0 \ f/x_1 \ f/x_3]$$

coincidiendo con el resultado anterior.

Tenemos nuestras definiciones e identidades. Repasemos la derivada de una función vectorial con un argumento vectorial, ya que la matriz resultante es especial. Lo encontraremos con frecuencia en el aprendizaje profundo.

## Jacobianos y hessianos

La ecuación 8.2 definió la derivada de una función vectorial,  $f$ , con respecto a un vector,  $x$ :

$$\mathbf{J}_x = \frac{\mathbf{F}}{x} = \begin{matrix} \frac{f_0}{x_0} & \frac{f_0}{x_1} & \dots & \frac{f_0}{x_{m-1}} \\ \vdots & \vdots & & \vdots \\ \frac{f_{n-1}}{x_0} & \frac{f_{n-1}}{x_1} & \dots & \frac{f_{n-1}}{x_{m-1}} \end{matrix} \quad (8.14)$$

Esta derivada se conoce como matriz jacobiana,  $J$ , o simplemente jacobiana, y lo encontrará de vez en cuando en la literatura sobre aprendizaje profundo, especialmente durante las discusiones sobre el descenso de gradientes y otros algoritmos de optimización utilizados en los modelos de entrenamiento. El jacobiano a veces tiene un subíndice para indicar la variable con respecto a la cual se  $x$  si con respecto a encuentra; por ejemplo,  $J_x$ . Cuando el contexto es claro, a menudo descuidamos el subíndice.

En esta sección, analizaremos el jacobiano y su significado. Luego presentaremos otra matriz, la matriz de Hesse (o simplemente la de Hesse), que se basa en la jacobiana, y aprenderemos a usarla en problemas de optimización.

La esencia de esta sección es la siguiente: el jacobiano es la generalización de la primera derivada y el hessiano es la generalización de la segunda derivada.

Con respecto a los

jacobianos Vimos anteriormente que podemos pensar en la Ecuación 8.14 como una pila de vectores gradientes transpuestos (Ecuación 8.3):

$$\mathbf{J}_x = \begin{matrix} f_0(x) \\ f_1(x) \\ \vdots \\ f_{n-1}(x) \end{matrix}$$

Ver el jacobiano como una pila de gradientes nos da una pista de lo que representa. Recuerde, el gradiente de un campo escalar, una función que acepta un argumento vectorial y devuelve un escalar, apunta en la dirección del cambio máximo en la función. De manera similar, el jacobiano nos brinda información sobre cómo se comporta la función vectorial en las proximidades de algún punto,  $x_p$ . El jacobiano es para funciones de vectores con valores vectoriales lo que el gradiente es para funciones de vectores con valores escalares; nos informa cómo cambia la función ante un pequeño cambio en la posición de  $x_p$ .

Una forma de pensar en el jacobiano es como una generalización del más específico. Casos específicos que encontramos en el capítulo 7. La tabla 8-2 muestra la relación entre la función y lo que mide su derivada.

**Tabla 8-2: La relación entre Jacobianos, gradientes y pendientes**

Función derivada	
f(x)	f/ x, matriz jacobiana
f(x)	f/ x, vector degradado
f(x)	df/dx, pendiente

La matriz jacobiana es la más general de las tres. Si limitamos el función a un escalar, entonces la matriz jacobiana se convierte en el vector gradiente (vector de fila en diseño de numerador). Si limitamos la función y el argumento a los escalares, el gradiente se convierte en la pendiente. En cierto sentido, todos ellos indican la Lo mismo: cómo cambia la función alrededor de un punto en el espacio.

El jacobiano tiene muchos usos. Presentaré dos ejemplos. El primero es de sistemas de ecuaciones diferenciales. El segundo utiliza el método de Newton para encontrar las raíces de una función vectorial. Veremos a los jacobianos nuevamente cuando analicemos la propagación hacia atrás, ya que requiere calcular las derivadas de una función con valores vectoriales con respecto a un vector.

#### Ecuaciones diferenciales autónomas

Una ecuación diferencial combina derivadas y valores de funciones en una ecuación. Las ecuaciones diferenciales aparecen en todas partes en física e ingeniería. Nuestro ejemplo proviene de la teoría de los sistemas autónomos, que son ecuaciones diferenciales donde la variable independiente no aparece en la lado derecho. Por ejemplo, si el sistema consta de valores de la función y primeras derivadas con respecto al tiempo, t, no hay t explícito en las ecuaciones que gobiernan el sistema.

El párrafo anterior es sólo para antecedentes; no necesitas memorizar orizarlo. Trabajar con sistemas de ecuaciones diferenciales autónomas conduce en última instancia al jacobiano, que es nuestro objetivo. Podemos ver el sistema como un función con valores vectoriales, y usaremos el jacobiano para caracterizar la función crítica puntos de ese sistema (los puntos donde la derivada es cero). Nosotros trabajamos con puntos críticos de funciones en el Capítulo 7.

Por ejemplo, exploremos el siguiente sistema de ecuaciones:

$$\frac{dx}{dt} = 4x - 2xy$$

$$\frac{dy}{dt} = 2y + xy - 2y^2$$

Este sistema incluye dos funciones, x(t) e y(t), y están acopladas de modo que la tasa de cambio de x(t) depende del valor de x y del valor de y, y viceversa.

Veremos el sistema como una función única con valores vectoriales:

$$f(x) = [f_0 \ f_1] = [4x_0^2 - 2x_1 \ x_0 x_1 - 2x_1^2], \quad x = [x_0 \ x_1] \quad (8.15)$$

donde reemplazamos  $x$  con  $x_0$  y  $y$  con  $x_1$ .

El sistema que representa  $f$  tiene puntos críticos en ubicaciones donde  $f = 0$ , siendo  $0$  el vector cero de dimensión  $2 \times 1$ . Los puntos críticos son

$$c_0 = [0 \ 0], \quad c_1 = [0 \ 1], \quad c_2 = [2 \ 2] \quad (8.16)$$

donde la sustitución en  $f$  muestra que cada punto devuelve el vector cero. Por el momento, supongamos que nos dieron los puntos críticos y ahora queremos caracterizarlos.

Para caracterizar un punto crítico, necesitaremos la matriz jacobiana que genera  $f$ :

$$J = \begin{bmatrix} \frac{f_0}{x_0} & \frac{f_0}{x_1} \\ \frac{f_1}{x_0} & \frac{f_1}{x_1} \end{bmatrix} = \begin{bmatrix} 4 & -2x_0 \\ -2x_1 & 2 + x_0 - 4x_1 \end{bmatrix} \quad (8.17)$$

Dado que el jacobiano describe cómo se comporta una función en las proximidades de un punto, podemos usarlo para caracterizar los puntos críticos. En el capítulo 7 usamos la derivada para decirnos si un punto era un mínimo o un máximo de una función. Para el jacobiano, utilizamos los valores propios de  $J$  de la misma manera para hablar sobre el tipo y la estabilidad de los puntos críticos.

Primero, encontremos el jacobiano en cada uno de los puntos críticos:

$$J_{x=c_0} = [4 \ 0 \ 2], \quad J_{x=c_1} = [2 \ 0 \ -2], \quad J_{x=c_2} = [0 \ -4 \ 2 \ -4]$$

Podemos usar NumPy para obtener los valores propios de los jacobianos:

---

```
>>> importar numpy
como np >>> np.linalg.eig([[4,0],
[0,2]])[0]
matriz([4., 2.]) >>> np.linalg.eig([[2,0],
[1,-2]])[0]
matriz([-2., 2.]) >>> np.linalg.eig([[0,-4],
[2,-4]])[0] matriz([-2.+2.j, -2.-2.j])
```

---

Encontramos `np.linalg.eig` en el Capítulo 6. Los valores propios son los primeros valores que devuelve `eig`, de ahí el subíndice `[0]` de la llamada a la función.

Para los puntos críticos de un sistema de ecuaciones diferenciales autónomas, la Los valores propios indican el tipo y la estabilidad de los puntos. Si ambos valores propios son reales y tienen el mismo signo, el punto crítico es un nodo. Si los valores propios

son menores que cero, el nodo es estable; de lo contrario, es inestable. Puedes pensar en un nodo estable como un pozo; si estás cerca, caerás en él. Un nodo inestable es como una colina; Si te alejas de la cima, el punto crítico, te caerás.

El primer punto crítico,  $c_0$ , tiene valores propios reales positivos; por lo tanto, representa un nodo inestable.

Si los valores propios del jacobiano son reales pero de signos opuestos, el punto crítico es un punto de silla. Hablamos de los puntos de silla en el Capítulo 7. Un punto de silla es, en última instancia, inestable, pero en dos dimensiones, hay una dirección en la que puedes "caer" en la silla y una dirección en la que puedes "caerte" de la silla. Algunos investigadores creen que la mayoría de los mínimos encontrados al entrenar redes neuronales profundas son en realidad puntos de silla de la función de pérdida. Vemos que el punto crítico  $c_1$  es un punto silla, ya que los valores propios son reales con signos opuestos.

Finalmente, los valores propios de  $c_2$  son complejos. Los valores propios complejos indican una espiral (también llamada foco). Si la parte real de los valores propios es menor que cero, la espiral es estable; de lo contrario, es inestable. Como los valores propios son conjugados complejos entre sí, los signos de las partes reales deben ser los mismos; uno no puede ser positivo mientras el otro es negativo. Para  $c_2$ , las partes reales son negativas, por lo que  $c_2$  indica una espiral estable.

### El Método de

Newton I presentó los puntos críticos de la Ecuación 8.15 por decreto. El sistema es lo suficientemente sencillo como para que podamos resolver los puntos críticos algebraicamente, pero en general ese no es el caso. Un método clásico para encontrar las raíces de una función (los lugares donde devuelve cero) se conoce como método de Newton. Este es un método iterativo que utiliza la primera derivada y una estimación inicial para concentrarse en la raíz. Veamos el método en una dimensión y luego ampliémoslo a dos. Veremos que pasar a dos o más dimensiones requiere el uso del jacobiano.

Usemos el método de Newton para encontrar la raíz cuadrada de 2. Para hacer eso, necesitamos una ecuación tal que  $f(\sqrt{2}) = 0$ . Un momento de reflexión nos da una: .  
 $f(x) = 2 - x^2$  Claramente, cuando  $x = \sqrt{2}$ ,  $f(x) = 0$ .

La ecuación rectora del método de Newton en una dimensión es

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (8.18)$$

donde  $x_0$  es una estimación inicial de la solución.

Sustituimos  $x_0$  por  $x_n$  en el lado derecho de la ecuación 8.18 para encontrar  $x_1$  . Luego repetimos usando  $x_1$  en el lado derecho para obtener  $x_2$  , y así sucesivamente hasta que veamos pocos cambios en  $x_n$  . En ese momento, si nuestra suposición inicial es razonable, tenemos el valor que buscamos. El método de Newton converge rápidamente, por lo que para ejemplos típicos, solo necesitamos un puñado de iteraciones. Por supuesto, tenemos potentes ordenadores a nuestro alcance, por lo que los usaremos en lugar de trabajar a mano. El código Python que necesitamos se encuentra en el Listado 8-1.

---

```

importar numpy como
np def f(x):
    devolver 2.0 - x*x def
d(x):
    devolver -2.0*x

x = 1,0
para i en el rango(5): x
    = x - f(x)/d(x) print("%2d:
    %0.16f" % (i+1,x))
print("NumPy dice sqrt(2) = %0.16f para una desviación de %0.16f" % (np.sqrt(2),
    np.abs(np.sqrt(2)-x)))

```

---

Listado 8-1: Hallando  $\sqrt{2}$  mediante el método de Newton

El Listado 8-1 define dos funciones. El primero,  $f(x)$ , devuelve el valor de la función para una  $x$  determinada. El segundo,  $d(x)$ , devuelve la derivada en  $x$ . Si  $f(x) = 2 - x$ ,  $f'(x) = -2x$ .

entonces  $f$  Nuestra estimación inicial es  $x = 1,0$ . Repetimos la ecuación 8.18 cinco veces, imprimiendo la estimación actual de la raíz cuadrada de 2 cada vez. Finalmente, usamos NumPy para calcular el valor real y ver qué tan lejos estamos de él.

La ejecución del Listado 8-1 produce

---

```

1: 1,5000000000000000
2: 1,4166666666666667 3:
1,4142156862745099 4:
1,4142135623746899 5:
1,4142135623730951

```

---

NumPy dice  $\sqrt{2} = 1,4142135623730951$  para una  
desviación de 0,0000000000000000

---

lo cual es impresionante; obtenemos  $\sqrt{2}$  a 16 decimales en sólo cinco iteraciones.

Para extender el método de Newton a funciones de vectores con valores vectoriales, como la ecuación 8.15, reemplazamos el recíproco de la derivada por el inverso del jacobiano. ¿Por qué lo contrario? Recuerde, para una matriz diagonal, la inversa es el recíproco de los elementos diagonales. Si vemos la derivada escalar como una matriz de  $1 \times 1$ , entonces el recíproco y el inverso son iguales. La ecuación 8.18 ya utiliza el inverso del jacobiano, aunque uno para una matriz de  $1 \times 1$ .

Por lo tanto, iteraremos

$$x_{n+1} = x_n - J^{-1} \frac{f(x_n)}{x=x_n} \quad (8.19)$$

para un valor inicial adecuado,  $x_0$ , y el inverso del jacobiano evaluado en  $x_n$ .

Usemos el método de Newton para encontrar los puntos críticos de la ecuación 8.15.

Antes de que podamos escribir código Python, necesitamos la inversa del jacobiano, la ecuación 8.17. La inversa de una matriz de  $2 \times 2$ ,

$$A = [abcd]$$

es

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

suponiendo que el determinante no es cero. El determinante de A es  $ad - bc$ . Por lo tanto, la inversa de la ecuación 8.17 es

$$J^{-1} = \frac{1}{(4 - 2x_1)(2 + x_0 - 4x_1) + 2x_0 x_1} \begin{bmatrix} 2x_0 & 0 \\ 0 & 2x_1 \end{bmatrix}$$

Ahora podemos escribir nuestro código. El resultado es el Listado 8-2.

---

importar numpy como np

```
def f(x):
    x0,x1 = x[0,0],x[1,0] return
    np.array([4*x0-2*x0*x1],[2*x1+x0*x1-2*x1**2])

def JI(x):
    x0,x1 = x[0,0],x[1,0] d =
    (4-2*x1)*(2-x0-4*x1)+2*x0*x1 retorno ( 1/
    d)*np.array([[2-x0-4*x1,2*x0],[-x1,4-2*x0]])

    x0 = float(entrada("x0: ")) x1 =
    float(entrada("x1: ")) x =
    np.array([[x0],[x1]])

    norte = 20
    para i en rango(N): x
    = x - JI(x) @ f(x) if (i > (N-10)):
        print("%4d: (%0.8f,
        %0.8f)" % (yo, x[0,0],x[1,0]))
```

---

Listado 8-2: El método de Newton en 2D

El Listado 8-2 hace eco del Listado 8-1 para el caso 1D. Tenemos  $f(x)$  para calcular el valor de la función para un vector de entrada dado y  $J_I(x)$  para darnos el valor del jacobiano inverso en  $x$ . Observe que  $f(x)$  devuelve un vector columna y  $J_I(x)$  devuelve una matriz de  $2 \times 2$ .

El código primero solicita al usuario sus conjeturas iniciales,  $x_0$  y  $x_1$ . Estos se forman en el vector inicial,  $x$ . Tenga en cuenta que formamos  $x$  explícitamente como un vector columna .

La implementación de la Ecuación 8.19 viene a continuación . El Jacobiano inverso es una matriz de  $2 \times 2$  que multiplicamos a la derecha por el valor de la función, un vector de columna de  $2 \times 1$ , usando el operador de multiplicación de matrices de NumPy, @. El resultado es un vector de columna de  $2 \times 1$  restado del valor actual de  $x$ , que a su vez es un vector de columna de  $2 \times 1$ . Si el ciclo está dentro de las 10 iteraciones posteriores a su finalización, el valor actual se imprime en la consola.

¿Funciona el Listado 8-2? Ejecutémoslo y veamos si podemos encontrar conjeturas iniciales que conduzcan a cada uno de los puntos críticos (Ecuación 8.16). Para una suposición inicial de

---

$x_0 = [-1 2]$ , obtenemos

---

```

11: (0,00004807, -1,07511237)
12: (0,00001107, -0,61452262)
13: (0,00000188, -0,27403667)
14: (0,00000019, -0,07568702)
15: (0,00000 001, -0,00755378)
16: (0,00000000, -0,00008442)
17: (0,00000000, -0,00000001)
18: (0,00000000, -0,00000000)
19: (0,00000000, -0,00000000)

```

---

que es el primer punto crítico de la Ecuación 8.15. Para encontrar los dos puntos críticos restantes, debemos elegir nuestras suposiciones iniciales con cierto cuidado. Algunas conjeturas explotan, mientras que muchas conducen de nuevo al vector cero. Sin embargo, un poco de prueba y error nos da

$$[1 1] \rightarrow [0 1] \text{ y } [1.6 1.8] \rightarrow [2 2]$$

mostrando que el método de Newton puede encontrar los puntos críticos de la ecuación 8.15.

Comenzamos esta sección con un sistema de ecuaciones diferenciales que interpretamos como una función con valores vectoriales. Luego utilizamos el Jacobiano para caracterizar los puntos críticos de ese sistema. A continuación, utilizamos el Jacobiano por segunda vez para localizar los puntos críticos del sistema mediante el método de Newton. Podríamos hacer esto porque el Jacobiano es la generalización del gradiente a funciones con valores vectoriales, y el gradiente en sí es una generalización de la primera derivada de una función escalar. Como se mencionó anteriormente, volveremos a ver a los Jacobianos cuando analicemos la propagación hacia atrás en el Capítulo 10.

Con respecto a las

hessianas Si la matriz Jacobiana es como la primera derivada de una función de una variable, entonces la matriz de Hesse es como la segunda derivada. En este caso, estamos restringidos a campos escalares, funciones que devuelven un valor escalar para una entrada vectorial.

Comencemos con la definición y partamos de ahí. Para la función  $f(x)$ , el hessiano se define como

$$\begin{array}{c}
 \frac{\partial^2 f}{\partial x_0^2} & \frac{\partial^2 f}{\partial x_0 \partial x_1} & \frac{\partial^2 f}{\partial x_0 \partial x_{n-1}} \\
 H_f = & \frac{\partial^2 f}{\partial x_1 \partial x_0} & \frac{\partial^2 f}{\partial x_1 \partial x_{n-1}} \\
 & \frac{\partial^2 f}{\partial x_n \partial x_0} & \frac{\partial^2 f}{\partial x_n \partial x_1} \\
 & & \frac{\partial^2 f}{\partial x_n^2}
 \end{array} \quad (8.20)$$

$$\text{donde } x = [x_0 \ x_1 \ \dots \ x_{n-1}]$$

La ecuación 8.20 nos dice que la hessiana es una matriz cuadrada. Además, es una matriz simétrica que implica  $H = H^T$ .

El hessiano es el jacobiano del gradiente de un campo escalar:

$$H_f = J(\nabla f)$$

Veamos esto con un ejemplo. Considere esta función:

$$f(x) = 2x_0^2 + x_0 x_2 + 3x_1 x_2 - x_1^2$$

Si usamos directamente la definición de hessiano en la ecuación 8.20, vemos  $\frac{\partial^2 f}{\partial x_0^2} = 4$  porque  $f'_x(x_0) = 4x_0 + x_2$ . Cálculos similares nos dan el resto. de la matriz de Hesse:

$$\begin{matrix}
 4 & 0 & 1 \\
 0 & -2 & 3 \\
 1 & 3 & 0
 \end{matrix}$$

En este caso, el hessiano es constante, no una función de  $x$ , porque la potencia más alta de una variable en  $f(x)$  es 2.

El gradiente de  $f(x)$ , usando nuestra definición de vector columna, es

$$f = \begin{matrix} f_0 \\ f_1 \\ f_2 \end{matrix} = \begin{matrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{matrix} = \begin{matrix} 4x_0 + x_2 \\ 3x_2 - 2x_1 \\ x_0 + 3x_1 \end{matrix}$$

con el jacobiano del gradiente dando lo siguiente, que es idéntica a la matriz que encontramos mediante el uso directo de la Ecuación 8.20.

$$J(\nabla f) = \begin{matrix} f_0 \\ f_1 \\ f_2 \end{matrix} = \begin{matrix} \frac{\partial f_0}{\partial x_0} & \dots & \frac{\partial f_0}{\partial x_2} \\ \frac{\partial f_1}{\partial x_0} & \dots & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_0} & \dots & \frac{\partial f_2}{\partial x_2} \end{matrix} = \begin{matrix} 4 & 0 & 1 \\ 0 & -2 & 3 \\ 1 & 3 & 0 \end{matrix}$$

## Mínimos y máximos

Vimos en el capítulo 7 que podíamos usar la segunda derivada para probar si los puntos críticos de una función eran mínimos ( $f < 0$ ). Veremos en la siguiente "segundo" sección cómo los máximos ( $f > 0$ ) utilizar los puntos críticos en problemas de optimización. Por ahora, usemos el hessiano para encontrar puntos críticos considerando sus valores propios. Continuaremos con el ejemplo anterior. La matriz de Hesse es  $3 \times 3$ , lo que significa que hay tres (o menos) valores propios. Nuevamente, ahorraremos tiempo y usaremos NumPy para decirnos cuáles son:

---

```
>>> np.linalg.eig([[4,0,1],[0,-2,3],[1,3,0]])[0]
array([ 4.34211128, 1.86236874, -4.20448002])
```

---

Dos de los tres valores propios son positivos y uno es negativo. Si los tres fueran positivos, el punto crítico sería mínimo. Asimismo, si los tres fueran negativos el punto crítico sería un máximo. Observe que la etiqueta mínimo/máximo es lo opuesto al signo, al igual que en el caso de una sola variable. Sin embargo, si al menos un valor propio es positivo y otro negativo, como es el caso de nuestro ejemplo, el punto crítico es un punto de silla.

Parece natural preguntar si existe el hessiano de una función vectorial,  $\mathbf{f}(\mathbf{x})$ . Después de todo, podemos calcular el jacobiano de dicha función; Lo hicimos arriba para mostrar que el hessiano es el jacobiano del gradiente.

Es posible extender el Hessiano a una función con valores vectoriales. Sin embargo, el resultado ya no es una matriz, sino un tensor de orden 3. Para ver que esto es así, considere la definición de una función con valores vectoriales:

$$\begin{aligned} f(x) = & \\ & f_0(x) \\ & f_1(x) \\ & f_{m-1}(x) \end{aligned}$$

Podemos pensar en una función con valores vectoriales, un campo vectorial, como un vector de funciones escalares de un vector. Podríamos calcular el hessiano de cada una de las  $m$  funciones en  $f$  para obtener un vector de matrices,

$$\begin{aligned} H_f = & \\ & H_{f0} \\ & H_{f1} \\ & \vdots \\ & H_{fm-1} \end{aligned}$$

pero un vector de matrices es un objeto 3D. Piense en una imagen RGB: una matriz 3D compuesta por tres imágenes 2D, una para cada canal rojo, verde y azul. Por lo tanto, si bien es posible definir y calcular, el hessiano de una función con valores vectoriales está más allá de nuestro alcance actual.

## Optimización

En el aprendizaje profundo, es más probable que vea el hessiano en referencia a la optimización. Entrenar una red neuronal es, en una primera aproximación, una

Problema de optimización: el objetivo es encontrar ponderaciones y sesgos que conduzcan a un mínimo en el panorama de la función de pérdida.

En el Capítulo 7 vimos que el gradiente proporciona información sobre cómo avanzar hacia un mínimo. Un algoritmo de optimización, como el descenso de gradiente, tema del Capítulo 11, utiliza el gradiente como guía. Como el gradiente es una primera derivada de la función de pérdida, los algoritmos basados únicamente en el gradiente se conocen como métodos de optimización de primer orden.

El hessiano proporciona información más allá del gradiente. Como segunda derivada, el Hessiano contiene información sobre cómo está cambiando el gradiente del paisaje de pérdidas, es decir, su curvatura. Una analogía de la física podría ayudar aquí. El movimiento de una partícula en una dimensión se describe mediante alguna función del tiempo,  $x(t)$ . La primera derivada, la velocidad, es  $dx/dt = v(t)$ . La velocidad nos dice qué tan rápido cambia la posición en el tiempo. Sin embargo, la velocidad puede cambiar con el tiempo, por lo que su derivada,  $dv/dt = a(t)$ , es la aceleración.

Y, si la velocidad es la primera derivada de la posición, entonces la aceleración es la segunda, la función d, la  $2x/dt^2 = a(t)$ . De manera similar, la segunda derivada de la pérdida hessiana, proporciona información sobre cómo está cambiando el gradiente. Los algoritmos de optimización que utilizan el Hessiano, o una aproximación del mismo, se conocen como métodos de optimización de segundo orden.

Comencemos con un ejemplo en una dimensión. Tenemos una función,  $f(x)$ , y actualmente estamos en algún  $x_0$ . Queremos pasar de esta posición a una nueva posición la posición, más cercana a un mínimo de  $f(x)$ . Un algoritmo de primer orden utilizará  $x_1$ , el gradiente, aquí la derivada, como guía, ya que sabemos que movernos en la dirección opuesta a la derivada nos moverá hacia un valor de función más bajo. Por lo tanto, para algún tamaño de paso, llámelo ( $\eta$ ), podemos escribir

$$x_1 = x_0 - f'(x_0)$$

Esto nos moverá de  $x_0$  hacia  $x_1$ , que está más cerca del mínimo de  $f(x)$ , suponiendo que el mínimo existe.

La ecuación anterior tiene sentido, entonces ¿por qué pensar en un método de segundo orden? El método de segundo orden entra en juego cuando pasamos de  $f(x)$  a  $f(x)$ . Ahora tenemos un gradiente, no sólo una derivada, y el panorama de  $f(x)$  alrededor de algún punto puede ser más complejo. La forma general de gradiente, el descenso es

$$x_1 = x_0 - f(x_0)$$

pero la información en hessiano puede resultar útil. Para ver cómo, primero debemos introducir la idea de una expansión en serie de Taylor, una forma de aproximar una función arbitraria como la suma de una serie de términos. Usamos series de Taylor con frecuencia en física e ingeniería para simplificar funciones complejas en las proximidades de un punto específico. También las usamos a menudo para calcular valores de funciones trascendentales (funciones que no se pueden escribir como un conjunto finito de operaciones de álgebra básica). Por ejemplo, es probable que cuando usas  $\cos(x)$  en un lenguaje de programación, el resultado se genere mediante una expansión en serie de Taylor.

con una cantidad suficiente de términos para obtener el coseno con una precisión de punto flotante de 32 o 64 bits:

$$= \approx 1 + \left(2 \frac{\cos 1.4}{\sin 1.4} (x) \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{k!}\right) - \frac{6x}{8x + 6! 8!}$$

En general, para aproximar una función,  $f(x)$ , en las proximidades de un punto,  $x = a$ , la expansión en serie de Taylor es

$$f(x) = (x - a) \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}$$

donde  $f^{(k)}(a)$  es la k-ésima derivada de  $f(x)$  evaluada en el punto  $a$ .

Una aproximación lineal de  $f(x)$  alrededor de  $x = a$  es

$$f(x) \approx f(a) + f'(a)(x - a)$$

mientras que una aproximación cuadrática de  $f(x)$  se convierte en

$$1 f(x) \approx f(a) + f'(a)(x - a) + f''(a)(x - a)^2 \quad (8.21)$$

donde vemos la aproximación lineal usando la primera derivada y la cuadrática usando la primera y segunda derivada de  $f(x)$ . Un algoritmo de optimización de primer orden utiliza la aproximación lineal, mientras que uno de segundo orden utiliza la aproximación cuadrática.

Pasar de una función escalar de un escalar,  $f(x)$ , a una función escalar de un vector,  $f(x)$ , cambia la primera derivada a un gradiente y la segunda derivada a la matriz de Hesse,

$$f(x) \approx f(a) + (x - a)^T f(a) + \frac{1}{2} (x - a)^T Hf(a)(x - a)$$

siendo  $Hf(a)$  la matriz de Hesse para  $f(x)$  evaluada en el punto  $a$ . El orden de los productos cambia para que las dimensiones funcionen correctamente, ya que ahora tenemos vectores y una matriz con la que trabajar.

Por ejemplo, si  $x$  tiene  $n$  elementos, entonces  $f(a)$  es un escalar; el gradiente en  $a$  es un vector columna de  $n$  elementos que multiplica  $(x - a)^T$ , un vector fila de  $n$  elementos, que produce un escalar; y el último término es  $1 \times n$  por  $n \times n$  por  $n \times 1$ , lo que lleva a  $1 \times n$  por  $n \times 1$ , que también es un escalar.

Para utilizar las expansiones de la serie de Taylor para la optimización, encontrar la mini-suma de  $f$ , podemos usar el método de Newton de manera muy similar a como se usa en la ecuación 8.18. Primero, reescribimos la ecuación 8.21 para cambiar nuestro punto de vista a uno de desplazamiento ( $\Delta x$ ) desde una posición actual ( $x$ ). La ecuación 8.21 entonces se convierte en

$$1 f(x + \Delta x) \approx f(x) + f'(x)\Delta x + f''(x)(\Delta x)^2 \quad (8.22)$$

La ecuación 8.22 es una parábola en  $\Delta x$ , y la estamos usando como sustituto de la forma más compleja de  $f$  en la región de  $x + \Delta x$ . Para encontrar el mínimo de la ecuación 8.22, tomamos la derivada y la ponemos en cero, luego resolvemos para  $\Delta x$ .

La derivada da

$$\frac{d}{d(\Delta x)} f(x + \Delta x) \approx f'(x) + f''(x)\Delta x$$

que, si se pone a cero, conduce a

$$\Delta x = -\frac{f'(x)}{f''(x)} \quad (8.23)$$

La ecuación 8.23 nos indica el desplazamiento desde una posición actual,  $x$ , que llevaría al mínimo de  $f(x)$  si  $f(x)$  fuera en realidad una parábola. En realidad,  $f(x)$  no es una parábola, por lo que  $\Delta x$  de la ecuación 8.23 no es el desplazamiento real del mínimo de  $f(x)$ . Sin embargo, dado que la expansión en serie de Taylor utilizó la pendiente real,  $f'(x)$ , y la curvatura,  $f''(x)$ , de  $f(x)$  en  $x$ , el desplazamiento de la ecuación 8.23 es una mejor estimación del mínimo real de  $f(x)$  que la aproximación lineal, asumiendo que hay un mínimo.

Si pasamos de  $x$  a  $x + \Delta x$ , no hay razón por la que no podamos usar la ecuación 8.23 por segunda vez, llamando  $x$  a la nueva posición. Pensar así lleva a una ecuación que podemos iterar:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \quad (8.24)$$

para  $x_0$ , algún punto de partida inicial.

Podemos resolver todo lo anterior para funciones escalares con argumentos vectoriales,  $f(x)$ , que son el tipo que encontramos con mayor frecuencia en el aprendizaje profundo a través de la función de pérdida. La ecuación 8.24 se convierte en

$$x_{n+1} = x_n - H_f^{-1}_{x_n} f(x_n)$$

donde el recíproco de la segunda derivada se convierte en el inverso de la matriz de Hesse evaluada en  $x_n$ .

¡Excelente! Tenemos un algoritmo que podemos usar para encontrar rápidamente el mínimo de una función como  $f(x)$ . Vimos anteriormente que el método de Newton converge rápidamente, por lo que usarlo para minimizar una función de pérdida también debería converger rápidamente, más rápido que el descenso de gradiente, que solo considera la primera derivada.

Si este es el caso, ¿por qué utilizamos el descenso de gradiente para entrenar redes neuronales? ¿Funciona en lugar del método de Newton?

Hay varias razones. Primero, no hemos discutido cuestiones que surgen de la aplicabilidad de Hesse, cuestiones relacionadas con que Hesse sea una matriz definida positiva. Una matriz simétrica es definida positiva si todos sus valores propios son positivos. Cerca de los puntos de silla, el hessiano podría no ser positivo definido, lo que puede hacer que la regla de actualización se aleje del mínimo. Como es de esperar con un algoritmo simple como el método de Newton, algunas variaciones intentan abordar problemas como este, pero incluso si los problemas con los valores propios

Se abordan los aspectos del Hessiano, la carga computacional que supone utilizar el Hessiano para actualizar los parámetros de la red es lo que detiene el algoritmo de Newton.

Cada vez que se actualizan los pesos y sesgos de la red, el Hessian cambia, lo que requiere que éste y su inverso se calculen nuevamente. Piense en la cantidad de minilotes utilizados durante el entrenamiento de la red. Incluso para un mini lote, hay  $k$  parámetros en la red, donde  $k$  fácilmente oscila entre millones e incluso miles de millones. La hessiana es una matriz definida positiva y simétrica  $ak \times k$ . La inversión del Hessiano normalmente utiliza la descomposición de Cholesky, que es más eficiente que otros métodos pero sigue siendo una notación ( $k$  indica que el uso de recursos del algoritmo escala como  $3^k$ ) algoritmo. La gran  $O$  el cubo del tamaño de la matriz en tiempo, memoria o ambos. Esto significa duplicar el El número de parámetros en la red aumenta el tiempo de cálculo para invertir el hessiano en un factor de 2, se requiere aproximadamente  $3 = 27$  veces más esfuerzo y cuadriplicar aproximadamente 4 veces más. Y esto sin mencionar el almacenamiento de  $k^3$  el Matriz de Hesse, todos los valores de punto flotante.  $3^3 = 8$  triplicando el número de parámetros = 64  $2^3 = 8$  elementos de

El cálculo necesario para utilizar el método de Newton incluso con redes profundas modestas es asombroso. Los métodos de optimización de primer orden basados en gradientes son todo lo que podemos utilizar para entrenar redes neuronales.

## NOTA

Esta afirmación quizá sea un poco prematura. Trabajos recientes en el área de la neuroevolución han demostrado que los algoritmos evolutivos pueden entrenar con éxito modelos profundos. Mi experimentación con técnicas de optimización de enjambres y redes neuronales también da crédito a este enfoque.

Que los métodos de primer orden funcionen tan bien parece, por ahora, un accidente muy feliz.

## Algunos ejemplos de derivados de cálculo matricial

Concluimos el capítulo con algunos ejemplos similares a los tipos de derivados que encontramos comúnmente en el aprendizaje profundo.

Derivada de operaciones por elementos Comencemos

con la derivada de operaciones por elementos, que incluye cosas como sumar dos vectores.  
Considerar

$$\begin{array}{ccc} f_0 & & a_0 + b_0 \\ f = a + b = & f_1 & = a_1 + b_1 \\ & & \\ & f_{n-1} & = a_{n-1} + b_{n-1} \end{array}$$

que es la simple suma de dos vectores, elemento por elemento.  
 ¿Cómo es  $f/a$ , el jacobiano de  $f$ ? De la definición tenemos

$$\frac{F}{a} = \begin{matrix} \frac{f_0}{a_0} & \frac{f_0}{a_1} & \frac{f_0}{a_{n-1}} \\ \frac{f_1}{a_0} & \frac{f_1}{a_1} & \frac{f_1}{a_{n-1}} \\ \vdots & \vdots & \vdots \\ \frac{f_{n-1}}{a_0} & \frac{f_{n-1}}{a_1} & \frac{f_{n-1}}{a_{n-1}} \end{matrix}$$

pero  $f_0$  sólo depende de  $a_0$ , mientras que  $f_1$  depende de  $a_1$ , y así sucesivamente. Por lo tanto, todas las derivadas  $f_i/a_j$  para  $i \neq j$  son cero. Esto elimina todos los elementos fuera de la diagonal de la matriz, dejando

$$\frac{F}{a} = \begin{matrix} \frac{f_0}{a_0} & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{f_1}{a_1} & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{f_{n-1}}{a_{n-1}} & 0 & 0 & 1 \end{matrix} = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} = \text{yo}$$

ya que  $f_i/a_i = 1$  para todo  $i$ . De manera similar,  $f/b = I$  también. Además, si pasamos de suma a resta,  $f/a = I$ , pero  $f/b = -I$ .

Si el operador es una multiplicación por elementos de  $a$  y  $b$ ,  $f = a \cdot b$ , entonces obtenemos lo siguiente, donde la notación  $\text{diag}(x)$  significa los  $n$  elementos del vector  $x$  a lo largo de la diagonal de una matriz  $n \times n$  que es cero en otros lugares.

$$\begin{aligned} \frac{F}{a} &= \begin{matrix} \frac{(a_0 b_0)}{a_0} & 0 & 0 & b_0 & 0 & 0 \\ 0 & \frac{(a_1 b_1)}{a_1} & 0 & 0 & b_1 & 0 \\ 0 & 0 & \frac{(a_{n-1} b_{n-1})}{a_{n-1}} & 0 & 0 & b_{n-1} \end{matrix} = \text{diag}(b) \\ \\ \frac{F}{b} &= \begin{matrix} \frac{(a_0 b_0)}{b_0} & 0 & 0 & a_0 & 0 & 0 \\ 0 & \frac{(a_1 b_1)}{b_1} & 0 & 0 & a_1 & 0 \\ 0 & 0 & \frac{(a_{n-1} b_{n-1})}{b_{n-1}} & 0 & 0 & a_{n-1} \end{matrix} = \text{diag}(a) \end{aligned}$$

### Derivada de la función de activación

Encontremos la derivada de los pesos y el valor de sesgo para un solo nodo de una capa oculta en una red feedforward. Recuerde, las entradas al nodo son las salidas de la capa anterior,  $x$ , multiplicadas término por término por los pesos,  $w$ , y sumadas junto con el valor de sesgo,  $b$ , un escalar. El resultado, un escalar, es

se pasa a la función de activación para producir el valor de salida para el nodo. Aquí, estamos usando la unidad lineal rectificada (ReLU) que devuelve su argumento si el argumento es positivo. Si el argumento es negativo, ReLU devuelve cero. Podemos escribir este proceso como

$$y = \text{ReLU}(w \cdot x + b) \quad (8.25)$$

Para implementar la retropropagación, necesitamos las derivadas de Ecuación 8.25 con respecto a  $w$  y  $b$ . Veamos cómo encontrarlos.

Comenzamos considerando las partes de la ecuación 8.25. Por ejemplo, a partir de la ecuación 8.9, sabemos que la derivada del producto escalar con respecto a  $w$  es

$$\frac{(x \cdot w)}{w} = x$$

donde hemos aprovechado que el producto escalar es commutativo,  $w \cdot x = x \cdot w$ . Además, como  $b$  no depende de  $w$ , tenemos

$$\frac{(w \cdot x + b)}{w} = \frac{(w \cdot x)}{w} + \frac{b}{w} = x + 0 = x \quad (8.26)$$

¿Qué pasa con el derivado de ReLU? Por definición,

$$\text{ReLU}(z) = \max(0, z) = \{0, z \leq 0\} z, z > 0$$

implicando que

$$\frac{\partial}{\partial z} \text{ReLU}(z) = \{0, z \leq 0\} z, z > 0 \quad (8.27)$$

ya que  $z/z = 1$ .

Para encontrar las derivadas de la ecuación 8.25 con respecto a  $w$  y  $b$ , necesitamos la regla de la cadena y los resultados anteriores. Empecemos con  $w$ . La regla de la cadena nos dice cómo

$$\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z} \frac{\partial z}{\partial w}$$

con  $z = w \cdot x + b$  y  $y = \text{ReLU}(z)$ .

Sabemos  $y/z$ ; son los dos casos anteriores para ReLU, Ecuación 8.27. Entonces ahora tenemos

$$\frac{\partial y}{\partial w} = \begin{cases} \frac{\partial y}{\partial z} & z \leq 0 \\ 1 & z > 0 \end{cases}$$

y sabemos  $z/w = x$ ; es la ecuación 8.26. Por lo tanto, nuestro resultado final es

$$\frac{y}{w} = \{0, w \cdot x + b \leq 0 \mid b > 0\}$$

donde hemos reemplazado  $z$  con  $w \cdot x + b$ .

Seguimos prácticamente el mismo procedimiento para encontrar  $y/b$ , como

$$\frac{y}{wb} = \frac{y}{z} - \frac{z}{b}$$

pero  $y/z$  es 0 o 1, dependiendo del signo de  $z$ . Asimismo,  $z/b = 1$ , lo que lleva a

$$\frac{yb}{w} = \{0, w \cdot x + b \leq 0 \mid b > 0\}$$

## Resumen

En este denso capítulo, aprendimos sobre el cálculo matricial, incluido el trabajo con derivadas de funciones que involucran vectores y matrices. Trabajamos en las definiciones y discutimos algunas identidades. Luego introdujimos las matrices jacobiana y hessiana como análogas para la primera y segunda derivada y aprendimos cómo usarlas en problemas de optimización. Entrenar una red neuronal profunda es, fundamentalmente, un problema de optimización, por lo que la utilidad potencial del jacobiano y el hessiano es clara, incluso si este último no puede usarse fácilmente para redes neuronales grandes. Terminamos el capítulo con algunos ejemplos de derivados de expresiones que se encuentran en el aprendizaje profundo.

Con esto concluye la parte de matemáticas generales del libro. Bien ahora Centramos nuestra atención en utilizar lo que hemos aprendido para comprender el funcionamiento de las redes neuronales profundas. Comencemos con una discusión sobre cómo fluyen los datos a través de un modelo de red neuronal.

# 9

## DOS TRABAJOS INNERALES DEL FLUJO DE DATOS



En este capítulo, presentaré cómo los datos fluye a través de una red neuronal entrenada. En otras palabras, veremos cómo pasar de un vector de entrada o tensor a la salida, y el forma que toman los datos a lo largo del camino. Si ya está familiarizado con el funcionamiento de las redes neuronales, genial, pero si no, analizar cómo fluyen los datos de una capa a otra La capa le ayudará a comprender los procesos involucrados.

Primero, veremos cómo representamos datos en dos tipos diferentes de redes. Luego, trabajaremos a través de una red de avance tradicional para brindar nosotros mismos una base sólida. Veremos cómo la inferencia compacta con una Red neuronal puede ser en términos de código. Finalmente, seguiremos los datos a través de un Red neuronal convolucional mediante la introducción de capas convolucionales y de agrupación. El objetivo de este capítulo no es presentar cómo los kits de herramientas populares transmiten datos. alrededor. Los kits de herramientas son piezas de software altamente optimizadas y un conocimiento de tan bajo nivel no nos resulta útil en esta etapa. En cambio, el objetivo es ayudar ves cómo los datos fluyen de la entrada a la salida.

Representar datos Al final,

todo en el aprendizaje profundo tiene que ver con datos. Tenemos datos que utilizamos para crear un modelo, que probamos con más datos, lo que en última instancia nos permite hacer predicciones sobre aún más datos. Comenzaremos viendo cómo representamos datos en dos tipos de redes neuronales: redes neuronales tradicionales y redes convolucionales profundas.

### Redes neuronales tradicionales

Para una red neuronal tradicional u otros modelos clásicos de aprendizaje automático, la entrada es un vector de números, el vector de características. Los datos de entrenamiento son una colección de estos vectores, cada uno con una etiqueta asociada. (En este capítulo nos limitaremos al aprendizaje supervisado básico). Una colección de vectores de características se implementa convenientemente como una matriz, donde cada fila es un vector de características y el número de filas coincide con el número de muestras en el conjunto de datos. Como sabemos ahora, una computadora representa convenientemente una matriz usando una matriz 2D. Por lo tanto, cuando trabajemos con redes neuronales tradicionales u otros modelos clásicos (máquinas de vectores de soporte, bosques aleatorios, etc.), representaremos conjuntos de datos como matrices 2D.

Por ejemplo, el conjunto de datos del iris, que encontramos por primera vez en el Capítulo 6, tiene cuatro características en cada vector de características. Lo representamos como una matriz:

---

```
>>> importar numpy como
np >>> desde sklearn importar conjuntos
de datos >>> iris = datasets.load_iris()
>>> X = iris.datos[:5]
>>> X
matriz([[5.1, 3.5, 1.4, 0.2], [4.9, 3. , 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2], [5. , 3.6, 1.4, 0.2]])
```

---

```
>>> Y = iris.objetivo[:5]
```

---

Aquí, hemos mostrado las primeras cinco muestras como lo hicimos en el Capítulo 6. Las muestras anteriores son todas para la clase 0 (I. setosa). Para pasar este conocimiento al modelo, necesitamos un vector coincidente de etiquetas de clase;  $X[i]$  devuelve el vector de características para la muestra  $i$  e  $Y[i]$  devuelve la etiqueta de clase. La etiqueta de clase suele ser un número entero y cuenta desde cero para cada clase en el conjunto de datos. Algunos kits de herramientas prefieren etiquetas de clase codificadas en caliente, pero podemos crearlas fácilmente a partir de etiquetas de enteros más estándar.

Por lo tanto, un conjunto de datos tradicional utiliza matrices entre capas para contener pesos, siendo la entrada y salida de cada capa un vector. Esto es bastante sencillo. ¿Qué tal una red más moderna y profunda?

## Redes convolucionales profundas

Las redes profundas pueden usar vectores de características, especialmente si el modelo implementa convoluciones 1D, pero la mayoría de las veces, el objetivo de usar una red profunda es permitir que las capas convolucionales aprovechen las relaciones espaciales en los datos. Generalmente, esto significa que las entradas son imágenes, que representamos usando matrices 2D. Pero no siempre es necesario que la entrada sea una imagen.

El modelo ignora felizmente lo que representa la entrada; Sólo el diseñador del modelo lo sabe y decide la arquitectura basándose en ese conocimiento.

Para simplificar, asumiremos que las entradas son imágenes, ya que ya conocemos cómo funcionan las computadoras con imágenes, al menos en un nivel alto.

Una imagen en blanco y negro, o una con tonos de gris, conocida como imagen en escala de grises, utiliza un único número para representar la intensidad de cada píxel. Por lo tanto, una imagen en escala de grises consta de una única matriz representada en la computadora como una matriz 2D. Sin embargo, la mayoría de las imágenes que vemos en nuestras computadoras son imágenes en color, no en escala de grises. La mayoría del software representa el color de un píxel mediante tres números: la cantidad de rojo, la cantidad de verde y la cantidad de azul. Éste es el origen de la etiqueta RGB que se le da a las imágenes en color en una computadora. Hay muchas otras formas de representar colores, pero RGB es, con diferencia, la más común. La combinación de estos colores primarios permite a las computadoras mostrar millones de colores. Si cada píxel necesita tres números, entonces una imagen en color no es una única matriz 2D, sino tres matrices 2D, una para cada color.

Por ejemplo, en el Capítulo 4, cargamos una imagen en color desde sklearn. Veámoslo nuevamente para ver cómo está organizado en la memoria:

---

```
>>> desde sklearn.datasets importar load_sample_image
>>> china = load_sample_image('china.jpg') >>>
china.shape (427,
640, 3)
```

---

La imagen se devuelve como una matriz NumPy. Al preguntar por la forma de la matriz, se devuelve una tupla: (427, 640, 3). La matriz tiene tres dimensiones. El primero es la altura de la imagen, 427 píxeles. El segundo es el ancho de la imagen, 640 píxeles. El tercero es el número de bandas o canales, aquí tres porque es una imagen RGB. El primer canal es el componente rojo del color de cada píxel, el segundo el verde y el último el azul. Podemos ver cada canal como una imagen en escala de grises si queremos:

---

```
>>> desde PIL importar
imagen >>> Imagen.fromarray(china).show()
>>> Imagen.fromarray(china[:, :, 0]).show()
>>> Imagen.fromarray(china[:, :, 1]).show()
>>> Imagen.fromarray(china[:, :, 2]).show()
```

---

PIL se refiere a Pillow, la biblioteca de Python para trabajar con imágenes. Si aún no lo tienes instalado, esto lo instalará por ti:

---

almohada de instalación pip3

---

Cada imagen parece similar, pero si las colocas una al lado de la otra, notarás diferencias. Consulte la Figura 9-1. El efecto neto de cada imagen por canal crea el color real mostrado. Reemplace `china[:, :, 0]` con solo `china` para ver la imagen a todo color.



Figura 9-1: Canales de imágenes de China rojo (izquierda), verde (centro) y azul (derecha)

Las entradas a las redes profundas suelen ser multidimensionales. Si la entrada es una imagen en color, necesitamos usar un tensor 3D para contener la imagen. Sin embargo, aún no hemos terminado. Cada muestra de entrada al modelo es un tensor 3D, pero rara vez trabajamos con una sola muestra a la vez. Al entrenar una red profunda, utilizamos minibatches, conjuntos de muestras procesadas juntas para obtener una pérdida promedio. Esto implica otra dimensión más para el tensor de entrada, una que nos permite especificar qué miembro del minibatch queremos. Por lo tanto, la entrada es un diezor 4D:  $N \times H \times W \times C$ , siendo  $N$  el número de muestras en el minibatch,  $H$  la altura de cada imagen en el minibatch,  $W$  el ancho de cada imagen y  $C$  el número de canales. A veces escribimos esto en forma de tupla como  $(N, H, W, C)$ .

Echemos un vistazo a algunos datos reales destinados a una red profunda. Los datos son el conjunto de datos CIFAR-10. Es un conjunto de datos de referencia ampliamente utilizado y está disponible aquí: <https://www.cs.toronto.edu/~kriz/cifar.html>. Sin embargo, no es necesario descargar el conjunto de datos sin procesar. Hemos incluido versiones de NumPy con el código de este libro. Como se mencionó anteriormente, necesitamos dos matrices: una para las imágenes y otra para las etiquetas asociadas. Los encontrará en los archivos `cifar10_test_images.npy` y `cifar10_test_labels.npy`, respectivamente. Vamos a ver:

---

```
>>> imágenes = np.load("cifar10_test_images.npy") >>> etiquetas =
np.load("cifar10_test_labels.npy") >>> imágenes.shape (10000, 32, 32,
3) >>> etiquetas. forma
(10000,)
```

---

Observe que la matriz de imágenes tiene cuatro dimensiones. El primero es el número de imágenes en la matriz ( $N = 10.000$ ). El segundo y el tercero nos dicen que las imágenes son de  $32 \times 32$  píxeles. El último nos dice que hay tres canales, lo que implica que el conjunto de datos consta de imágenes en color. Tenga en cuenta que, en general, el número de

Los canales pueden referirse a cualquier colección de datos agrupados de esa manera; no es necesario que sea una imagen real. El vector de etiquetas también tiene 10.000 elementos. Estas son las etiquetas de clases, de las cuales hay 10 clases, una mezcla de animales y vehículos. Por ejemplo,

---

```
>>> etiquetas[123]
2
>>> Imagen.dearray(imágenes[123]).mostrar()
```

---

Esto indica que la imagen 123 es de clase 2 (pájaro) y que la etiqueta es correcta; la imagen mostrada debe ser la de un pájaro. Recuerde que, en NumPy, al solicitar un único índice se devuelve el subarreglo completo, por lo que imágenes[123] es equivalente a imágenes[123,:,:,:]. El método fromarray de la clase Image convierte una matriz NumPy en una imagen para que show pueda mostrarla.

Trabajar con minibatches significa que pasamos un subconjunto del conjunto de datos completo a través del modelo. Si nuestro modelo usa minibatches de 24, entonces la entrada a la red profunda, si usa CIFAR-10, es una matriz (24, 32, 32, 3): 24 imágenes, cada una de las cuales tiene 32 filas, 32 columnas y 3 canales. Veremos a continuación que la idea de canales no se limita a la entrada a una red profunda; también se aplica a la forma de los datos que se pasan entre capas.

Volveremos a los datos para redes profundas en breve. Pero por ahora, pasemos al tema más sencillo del flujo de datos en una red neuronal tradicional de alimentación directa.

## Flujo de datos en redes neuronales tradicionales

Como indicamos anteriormente, en una red neuronal tradicional, los pesos entre capas se almacenan como matrices. Si la capa  $i$  tiene  $n$  nodos y la capa  $i - 1$  tiene  $m$  salidas, entonces la matriz de pesos entre las dos capas,  $W_i$ , es una matriz  $n \times m$ .

Cuando esta matriz se multiplica a la derecha por el vector columna  $m \times 1$  de salidas de la capa  $i - 1$ , el resultado es una salida  $n \times 1$  que representa la entrada a los  $n$  nodos para la capa  $i$ . En concreto calculamos

$$a_i = (W_{i-1} \cdot b_i)$$

donde  $a_{i-1}$ , el vector  $m \times 1$  de salidas de la capa  $i - 1$ , multiplica  $W_i$  para producir un vector columna  $n \times 1$ . Agregamos  $b_i$ , los valores de sesgo para la capa  $i$ , a este vector y aplicamos la función de activación, a cada elemento del vector resultante,  $W_{i-1} \cdot b_i$ , para producir  $a_i$ , las activations para la capa  $i$ . Alimentamos las activations a la capa  $i + 1$  como salida de la capa  $i$ . Al utilizar matrices y vectores, las reglas de multiplicación de matrices calculan automáticamente todos los productos necesarios sin bucles explícitos en el código.

Veamos un ejemplo con una red neuronal simple. Generaremos un conjunto de datos aleatorio con dos características y luego dividiremos este conjunto de datos en grupos de entrenamiento y prueba. Usaremos sklearn para entrenar una red neuronal de avance simple en el conjunto de entrenamiento. La red tiene una única capa oculta con cinco nodos y utiliza una función de activación lineal rectificada (ReLU). Luego probaremos los entrenados.

red para ver qué tan bien aprendió y, lo más importante, observar las matrices de peso y los vectores de sesgo reales.

Para construir el conjunto de datos, seleccionaremos un conjunto de puntos en el espacio 2D que estén agrupados pero ligeramente superpuestos. Queremos que la red tenga que aprender algo que no sea completamente trivial. Aquí está el código:

---

```
desde sklearn.neural_network importe MLPClassifier
```

```
np.random.seed(8675309)
x0 = np.random.random(50)-0.3
y0 = np.random.random(50)+0.3
x1 = np.random.random(50)+0.3
y1 = np.random .random(50)-0.3
x = np.zeros((100,2))
x[:50,0] = x0; x[:50,1] = y0
x[50:,0] = x1; x[50:,1] = y1
y = np.array([0]*50+[1]*50)

idx = np.argsort(np.random.random(100))
x = x[idx]; y = y[idx]
x_train = x[:75]; x_test = x[75:]
y_train = y[:75]; y_prueba = y[75:]
```

---

Necesitamos la clase `MLPClassifier` de `sklearn`, así que la cargamos primero. Luego definimos un conjunto de datos 2D, `x`, que consta de dos nubes de 50 puntos cada una. Los puntos están distribuidos aleatoriamente ( $x_0, y_0$  y  $x_1, y_1$ ) pero centrados en  $(0,2, 0,8)$  y  $(0,8, 0,2)$ , respectivamente . Tenga en cuenta que configuramos la semilla de números aleatorios de NumPy en un valor fijo, por lo que cada ejecución produce el mismo conjunto de números que veremos a continuación. Siéntase libre de eliminar esta línea y experimentar qué tan bien se entrena la red para varias generaciones del conjunto de datos.

Sabemos que los primeros 50 puntos en `x` son de lo que llamaremos clase 0, y el . Los siguientes 50 puntos son de clase 1, por lo que definimos un vector de etiqueta, `y` . Finalmente, aleatorizamos el orden de los puntos en `x` , teniendo cuidado de alterar las etiquetas de la misma manera, y los dividimos en un conjunto de entrenamiento (`x_train`) y etiquetas (`y_train`) y un conjunto de prueba (`x_test`) y etiquetas (`y_test`). Guardamos el 75 por ciento de los datos para el entrenamiento y dejamos el 25 por ciento restante para las pruebas.

La Figura 9-2 muestra un gráfico del conjunto de datos completo, con cada característica en uno de los ejes. Los círculos corresponden a instancias de clase 0 y los cuadrados a instancias de clase 1. Existe una clara superposición entre las dos clases.

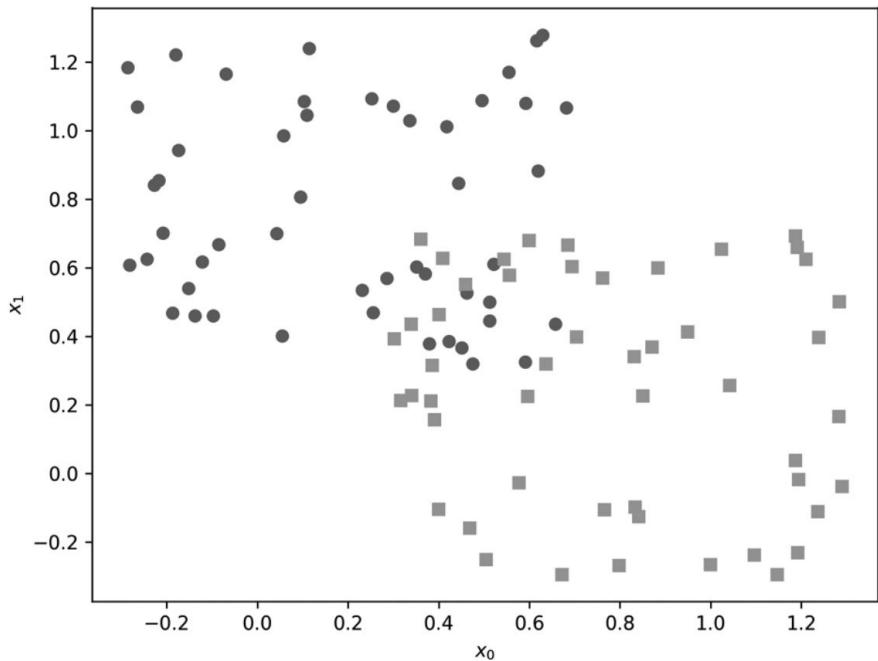


Figura 9-2: El conjunto de datos utilizado para entrenar la red neuronal, con las instancias de clase 0 mostradas como círculos y las instancias de clase 1 como cuadrados

Ahora estamos listos para entrenar el modelo. El kit de herramientas sklearn facilita nosotras, si utilizamos los valores predeterminados:

---

```

clf = MLPClassifier(hidden_layer_sizes=(5,))
clf.fit(tren_x, tren_y)

puntuación = clf.score(x_test, y_test)
print("Precisión del modelo en el conjunto de pruebas: %0.4f" % puntuación)

W0 = clf.coefs_[0].T b0 =
clf.intercepts_[0].reshape((5,1))
W1 = clf.coefs_[1].T
b1 = clf.intercepts_[1]

```

---

La capacitación implica la creación de una instancia de la clase modelo . Darse cuenta de Al usar los valores predeterminados, que incluyen el uso de una función de activación ReLU, solo necesitamos especificar la cantidad de nodos en las capas ocultas. Queremos una capa oculta con cinco nodos, por lo que pasamos la tupla (5,). El entrenamiento es una llamada única a la función de ajuste que pasa los datos de entrenamiento, x\_train, y las etiquetas asociadas, y\_train. Cuando esté completo, probamos el modelo calculando la precisión (puntuación) en el conjunto de prueba (x\_test, y\_test) y mostramos el resultado.

Las redes neuronales se inicializan aleatoriamente, pero debido a que fijamos la semilla de números aleatorios NumPy cuando generamos el conjunto de datos, y debido a que sklearn también usa el generador de números aleatorios NumPy, el resultado del entrenamiento de la red debería ser el mismo para cada ejecución del código. El modelo tiene una precisión del 92 por ciento en los datos de prueba . Esto es conveniente para nosotros, pero también nos preocupa: hay tantos kits de herramientas que utilizan NumPy internamente que las interacciones debidas a la corrección de la semilla de números aleatorios son probables, generalmente indeseables y quizás difíciles de detectar.

Finalmente estamos listos para obtener las matrices de peso y los vectores de sesgo de la red entrenada . Debido a que sklearn usa np.dot para la multiplicación de matrices, tomamos la transposición de las matrices de peso, W0 y W1, para obtenerlas en una forma que sea más fácil de seguir matemáticamente. Veremos precisamente por qué esto es necesario a continuación. Del mismo modo, b0, el vector de polarización de la capa oculta, es una matriz NumPy 1D, por lo que lo cambiamos a un vector de columna. El sesgo de la capa de salida, b1, es un escalar, ya que solo hay una salida para esta red, el valor que pasamos a la función sigmoidea para obtener la probabilidad de ser miembro de la clase 1.

Recorramos la red para ver la primera muestra de prueba. Para ahorrar espacio, solo mostraremos los primeros tres dígitos de los valores numéricos, pero nuestros cálculos utilizarán precisión total. La entrada a la red es

$$x = [0,252 \ 1,092]$$

Queremos que la red nos proporcione una salida que conduzca a la probabilidad de que esta entrada pertenezca a la clase 1.

Para obtener la salida de la capa oculta, multiplicamos x por la matriz de peso, W0 , sumamos el vector de polarización, b0 , y pasamos ese resultado a través de ReLU:

$$a0 = \text{ReLU}(W0 x + b)$$

$$\begin{aligned}
 & \begin{array}{ccc} 0,111 & 1,018 & -0,055 \\ 0,419 & -0,547 & 0,238 \\ 0,137 & 0,615 & -0,280 \\ \text{=ReLU} & -0,427 & -0,225 \end{array} \\
 & \quad [0,252 \ 1,092] + \begin{array}{c} -0,313 \\ 0,901 \end{array} \\
 & \begin{array}{c} 1,084 \\ -0,254 \\ 0,427 \\ \text{=ReLU} \end{array} \\
 & \quad \begin{array}{c} -0,667 \\ 0,187 \end{array} \\
 & \begin{array}{c} 1,084 \\ 0. \\ \text{=} \end{array} \\
 & \quad \begin{array}{c} 0,427 \\ 0. \end{array} \\
 & \quad 0,187
 \end{aligned}$$

La transición de capa oculta a salida usa la misma forma, con  $a_0$  en lugar de  $x$ , pero aquí no se aplica ReLU:

$$\begin{aligned}
 a_1 &= W_1 a_0 + b_1 \\
 &\quad \begin{array}{r} 1.084 \\ 0. \\ 0. \\ \hline 0,187 \end{array} \\
 &= [-0,383 \ 1,227 \ -0,938 \ 0,329 \ -0,638] \quad \begin{array}{r} 0,427 \\ 0. \\ \hline + 0,340 \end{array} \\
 &\quad \begin{array}{r} \\ \\ = -0,59575099 \end{array}
 \end{aligned}$$

Para obtener la probabilidad de salida final, utilizamos  $a_1$ , un valor escalar, como argumento de la función sigmoidea, también llamada función logística:

$$\text{sigmoide}(a_1) = 1 \frac{1}{1 + e^{-a_1}} = \frac{1}{1 + m10.59575099} = 0,3553164$$

Esto significa que la red ha asignado una probabilidad del 35,5 por ciento de que el valor de entrada sea miembro de la clase 1. El umbral habitual para la asignación de clases para un modelo binario es del 50 por ciento, por lo que la red asignaría  $x$  a la clase 0. Un vistazo a `y_test[ 0 ]` nos dice que la red es correcta en este caso:  $x$  es de clase 0.

## Flujo de datos en redes neuronales convolucionales

Vimos anteriormente cómo el flujo de datos a través de una red neuronal tradicional era una matemática sencilla de matriz-vector. Para rastrear el flujo de datos a través de una red neuronal convolucional (CNN), primero debemos aprender qué es la operación de convolución y cómo funciona. Específicamente, aprenderemos cómo pasar datos a través de capas convolucionales y de agrupación a una capa completamente conectada en la parte superior del modelo. Esta secuencia representa muchas arquitecturas CNN, al menos a nivel conceptual.

### Circunvolución

La convolución implica dos funciones y el deslizamiento de una sobre la otra. Si las funciones son  $f(x)$  y  $g(x)$ , la convolución se define como

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (9.1)$$

Afortunadamente para nosotros, estamos trabajando en un dominio discreto y la mayoría de las veces con entradas 2D, por lo que la integral en realidad no se usa, aunque sigue siendo una notación útil para la operación.

El efecto neto de la ecuación 9.1 es deslizar  $g(x)$  sobre  $f(x)$  para diferentes desplazamientos. Aclaremos usando un ejemplo discreto 1D.

## Convolución en una dimensión

La Figura 9-3 muestra un gráfico en la parte inferior y dos conjuntos de números etiquetados f y g en la parte superior.

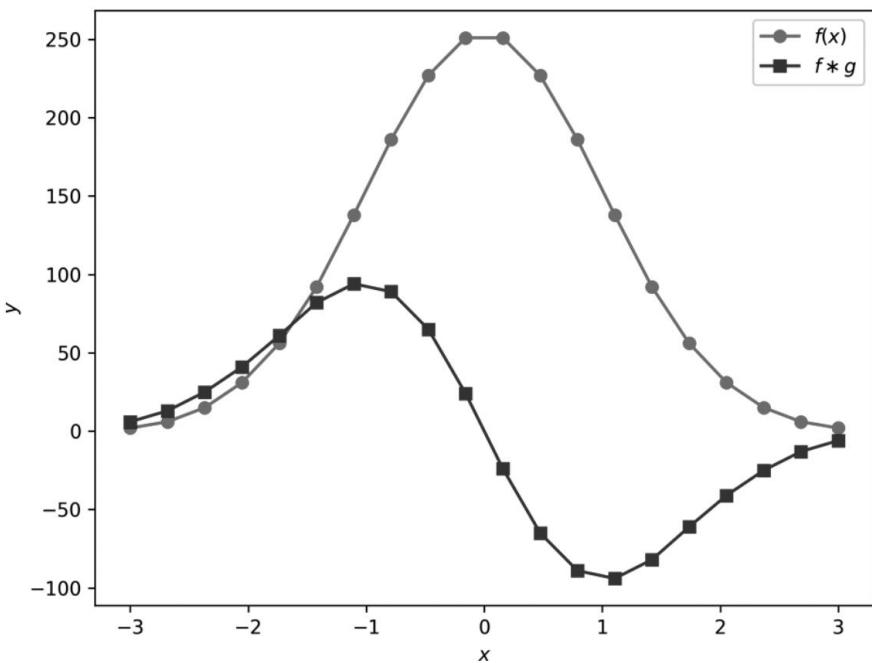
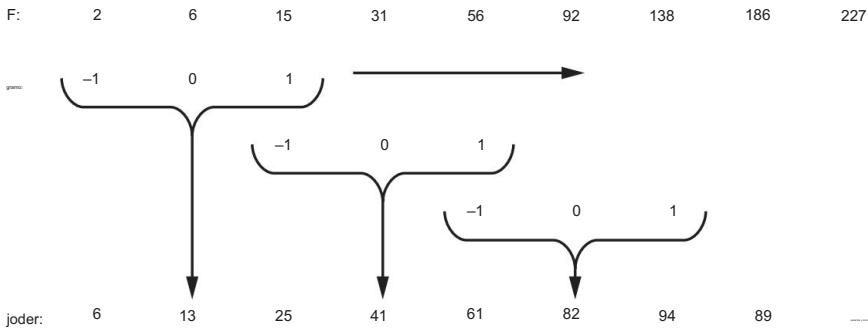


Figura 9-3: Una convolución discreta 1D

Comencemos con los números que se muestran en la parte superior de la Figura 9-3. La primera fila enumera los valores discretos de f. Debajo está g, una rampa lineal de tres elementos. La convolución alinea g con el borde izquierdo de f como se muestra. Multiplicamos los elementos correspondientes entre las dos matrices,

$$[2, 6, 15] \times [-1, 0, 1] = [-2, 0, 15]$$

y luego sumar los valores resultantes,

$$-2 + 0 + 15 = 13$$

para producir el valor que va en el elemento indicado de la salida, f g.

Para completar la convolución, g desliza un elemento hacia la derecha y el proceso se repite.

Tenga en cuenta que en la Figura 9-3, mostramos cada dos alineamientos de f y g para mayor claridad, por lo que parecerá como si g estuviera deslizando dos elementos hacia la derecha. En general, nos referimos a g como núcleo, el conjunto de valores que se deslizan sobre la entrada, f.

El gráfico en la parte inferior de la Figura 9-3 es  $f(x) = 255 \exp(-0.5x^2)$  para  $x$  en  $[-3, 3]$  en los puntos marcados con círculos. La operación mínima convierte la salida en un número entero para simplificar la discusión siguiente.

Los puntos cuadrados en la Figura 9-3 son el resultado de la convolución de  $f(x)$  con  $g(x) = [-1, 0, 1]$ .

Los puntos f y f g en la Figura 9-3 se generan mediante

---

```
x = np.linspace(-3,3,20)
f = (255*np.exp(-0.5*x**2)).astype("int32")
g = np.array([-1,0, 1])
fp= np.convolve(f,g[::-1], mode='mismo')
```

---

Este código requiere alguna explicación.

Primero, tenemos x, un vector que abarca  $[-3, 3]$  en 20 pasos; este vector genera f ( $f(x)$  arriba). Queremos que f sea de tipo entero, que es lo que hace astype por nosotros. A continuación, definimos g, la pequeña rampa lineal. Como veremos, la operación de convolución desliza g sobre los elementos de f para producir la salida.

La operación de convolución viene a continuación. Como se usa comúnmente la convolución, NumPy proporciona una función de convolución 1D, np.convolve. El primer argumento es f y el segundo es g. Explicaré en breve por qué agregamos  $[::-1]$  a g para revertirlo. También explicaré el significado de mode='same'. La salida de la convolución se almacena en fp.

La primera posición que se muestra en la parte superior de la Figura 9-3 completa el 13 en la salida. ¿De dónde viene el 6 a la izquierda del 13? La convolución tiene problemas en los bordes de f, donde el núcleo no cubre completamente la entrada. Para un núcleo de tres elementos, habrá un elemento de borde en cada extremo de f. Los núcleos suelen tener un número impar de valores, por lo que hay un elemento intermedio claro. Si g tuviera cinco elementos, habría dos elementos en cada extremo de f que g no cubriría.

Las funciones de convolución necesitan tomar una decisión sobre estos casos extremos. Una opción sería devolver solo la parte válida de la convolución, para ignorar los casos extremos. Si hubiéramos utilizado este enfoque, llamado convolución válida, la salida, yp, comenzaría con el elemento 13 y tendría dos longitudes menos que la entrada, y.

Otro enfoque es completar los valores faltantes en f con cero. Esto se conoce como relleno cero y normalmente lo usamos para hacer que la salida de una operación de convolución tenga el mismo tamaño que la entrada.

El uso de mode='same' con np.convolve selecciona el relleno de ceros. Esto explica el 6 a la izquierda del 13. Es lo que obtenemos al agregar un cero antes del 2 en f y aplicar el núcleo:

$$[0, 2, 6] \times [-1, 0, 1] = [0, 0, 6], 0 + 0 + 6 = 6$$

Si solo quisieramos los valores de salida válidos, habríamos usado mode='valid' en su lugar.

La llamada a np.convolve anterior no usó g. Pasamos g[::-1] en su lugar, el reverso de g. Hicimos esto para que np.convolve actuara como las convoluciones utilizadas en redes neuronales profundas. Desde una perspectiva matemática y de procesamiento de señales, la convolución utiliza el reverso del núcleo. La función np.convolve , por lo tanto, invierte el núcleo, lo que significa que debemos invertirlo de antemano para obtener el efecto que queremos. Para ser técnicos, si realizamos la operación que hemos llamado convolución sin invertir el núcleo, en realidad estamos realizando una correlación cruzada. Este problema rara vez surge en el aprendizaje profundo porque aprendemos los elementos del núcleo durante el entrenamiento y no los asignamos con anticipación. Con eso en mente, cualquier cambio del kernel por parte del proceso del kit de herramientas que implementa la operación de convolución no afectará el resultado, porque los valores aprendidos del kernel se aprendieron con ese cambio implementado. Asumiremos que en el futuro no hay inversión y, cuando sea necesario, invertiremos los núcleos que asignamos a las funciones NumPy y SciPy. Además, continuaremos usando el término convolución en este sentido de aprendizaje profundo sin voltear el kernel.

En general, la convolución con entradas discretas implica colocar el núcleo sobre la entrada comenzando por la izquierda, multiplicar los elementos coincidentes, sumar y poner el resultado en la salida en el punto donde coincide el centro del núcleo. Luego, el núcleo desliza un elemento hacia la derecha y el proceso se repite. Podemos extender la operación de convolución discreta a dos dimensiones. La mayoría de las CNN profundas modernas utilizan núcleos 2D, aunque también es posible utilizar núcleos 1D y 3D.

### Convolución en dos dimensiones

La convolución con un núcleo 2D requiere una matriz 2D. Las imágenes son matrices de valores 2D y la convolución es una operación común de procesamiento de imágenes. Por ejemplo, carguemos una imagen, la cara del mapache que vimos en el Capítulo 3, y modifiquela con una convolución 2D. Considera lo siguiente:

---

```
de scipy.signal importar convolve2d de
scipy.misc importar cara

img =
cara(Verdadero) img = img[:512,(img.shape[1]-612):(img.shape[1]-100)]

k = np.array([[1,0,0],[0,-8,0],[0,0,3]]) c =
convolve2d(img, k, mode='mismo')
```

---

Aquí, estamos usando la función SciPy convolve2d del módulo de señal .  
 Primero, cargamos la imagen del mapache y la subconfiguramos en una imagen de  $512 \times 512$  píxeles de la cara del mapache (img). A continuación, definimos un núcleo de  $3 \times 3$ , k. Por último, convolucionamos el núcleo, tal como está, con la imagen de la cara, almacenando el resultado en c. La palabra clave mode='same' rellena con ceros la imagen para manejar los casos extremos.  
 El código anterior conduce a

```
img[:8,:8]:
[[ 88 97 112 127 116 97 84 84] [ 62
 70 100 131 126 88 52 51] [ 41 46
 87 127 146 116 78 56] [ 42 45 76
 107 145 137 112 76] [ 58 59 69 79
 111 106 90 68] [ 74 73 68 60 72 74
 72 67] [ 92 87 75 63 57 74 91 93]
[105 97 85 74 60 79 102 110]]
```

k:

```
[[ 1 0 0] [ 0
 -8 0] [ 0 0
 3]]
```

c[1:8,1:8]:

```
[[-209 -382 -566 -511 -278 -69 -101]
 [-106 -379 -571 -638 -438 -284 -241]
 [-168 -391 -484 -673 -568 -480 -318]
 [-278 -357 -332 -493 -341 -242 -143]
 [-335 -304 -216 -265 -168 -165 -184]
 [-389 -307 -240 -197 -274 -396 -427]
 [-404 -331 -289 -215 -368 -476 -488]]
```

Aquí, mostramos la esquina superior de  $8 \times 8$  de la imagen y la porción válida de la convolución. Recuerde, la parte válida es la parte donde el kernel cubre completamente la matriz de entrada.

Para el núcleo y la imagen, la primera salida de convolución válida es -209. Matemáticamente, el primer paso es la multiplicación de elementos con el núcleo,

$$\begin{array}{r} 88 \ 97 \ 112 \\ 62 \ 70 \ 100 \\ 41 \ 46 \ 87 \end{array} \quad \begin{array}{r} 3 \ 0 \ 0 \\ 0 \ -8 \ 0 \\ 0 \ 0 \ 1 \end{array} = \begin{array}{r} 264 \ 0 \ 0 \\ 0 \ -560 \ 0 \ 0 \ 0 \\ 87 \end{array}$$

seguido de un resumen,

$$264 + 0 + 0 + 0 + (-560) + 0 + 0 + 0 + 87 = -209$$

Observe cómo el kernel utilizado no era k tal como lo definimos. En cambio, convolve2d volteó el núcleo de arriba a abajo y luego de izquierda a derecha antes de aplicarlo. El resto de c fluye al mover el núcleo una posición hacia la derecha.

y repetir la multiplicación y la suma. Al final de una fila, el kernel baja una posición y regresa a la izquierda, hasta que se haya procesado toda la imagen. Los kits de herramientas de aprendizaje profundo se refieren a este movimiento como zancada, y no es necesario que sea una posición o igual en las direcciones horizontal y vertical.

La Figura 9-4 muestra el efecto de la convolución.



Figura 9-4: Imagen original de la cara del mapache (izquierda) y resultado de la convolución (derecha)

Para crear la imagen, se desplazó hacia arriba, por lo que el valor mínimo fue cero y luego se dividió por el máximo para asignarlo a  $[0, 1]$ . Finalmente, el resultado se multiplicó por 255 y se mostró como una imagen en escala de grises. La imagen de la cara original está a la izquierda. La imagen convolucionada está a la derecha. La convolución de la imagen con el núcleo ha alterado la imagen, enfatizando algunas características y suprimiendo otras.

La convolución de núcleos con imágenes no es simplemente un ejercicio para ayudarnos a desentrañar comprender la operación de convolución. Es de profunda importancia en la formación de las CNN. Conceptualmente, una CNN consta de dos partes principales: un conjunto de convolución y otras capas a las que se les enseña a aprender una nueva representación de la entrada, y un clasificador de nivel superior al que se le enseña a usar la nueva representación para clasificar las entradas. Es el aprendizaje conjunto de la nueva representación y el clasificador lo que hace que las CNN sean tan poderosas. La clave para aprender una nueva representación de la entrada es el conjunto de núcleos de convolución aprendidos. La forma en que los núcleos alteran la entrada a medida que los datos fluyen a través de la CNN crea la nueva representación. El entrenamiento con descenso de gradiente y retropropagación le enseña a la red qué núcleos crear.

Ahora estamos en condiciones de seguir los datos a través de las capas convolucionales de una CNN. Vamos a ver.

### Capas convolucionales

Arriba, analizamos cómo las redes profundas pasan tensores de capa a capa y cómo el tensor generalmente tiene cuatro dimensiones,  $N \times H \times W \times C$ . Para seguir los datos a través de una capa convolucional, ignoraremos  $N$ , sabiendo que lo que discutir se aplica a cada muestra en el tensor. Esto nos deja con entradas a la capa convolucional que son  $H \times W \times C$ , un tensor 3D.

La salida de una capa convolucional es otro tensor 3D. La altura y el ancho de la salida dependen de los detalles de los núcleos de convolución y cómo decidimos manejar los bordes. Usaremos una convolución válida para ejemplos aquí, lo que significa que descartaremos partes de la entrada que el kernel no cubre del todo. Si el kernel es  $3 \times 3$ , la salida será dos menos en alto y ancho, uno menos por cada borde. Un grano de  $5 \times 5$  pierde cuatro de altura y ancho, dos menos por cada borde.

La capa convolucional utiliza conjuntos de filtros para lograr su objetivo. Un filtro es una pila de granos. Necesitamos un filtro para cada uno de los canales de salida deseados. El número de núcleos en la pila de cada filtro coincide con el número de canales en la entrada. Entonces, si la entrada tiene  $M$  canales y queremos  $N$  canales de salida usando  $K \times K$  kernels, necesitamos  $N$  filtros, cada uno de los cuales es una pila de granos  $MK \times K$ .

Además, tenemos un valor de sesgo para cada uno de los  $N$  filtros. Veremos a continuación cómo se utiliza el sesgo, pero ahora sabemos cuántos parámetros necesitamos. aprender a implementar una capa convolucional con  $M$  canales de entrada,  $K \times K$  núcleos y  $N$  salidas. Es  $K \times K \times M \times N$  para  $N$  filtros con  $K \times K \times M$  parámetros cada uno, más  $N$  términos de sesgo, uno por filtro.

Hagamos todo esto concreto. Tenemos una capa convolucional. La entrada a la capa es un tensor  $(H,W,C) = (5,5,2)$ , es decir, una altura y un ancho. de cinco y dos canales. Usaremos un kernel  $3 \times 3$  con convolución válida, entonces la salida en alto y ancho es  $3 \times 3$  de la entrada  $5 \times 5$ . Llegamos a seleccionar el número de canales de salida. Usemos tres. Por lo tanto, necesitamos usar convolución y kernels para asignar una entrada  $(5,5,2)$  a una salida  $(3,3,3)$ . De lo que discutimos anteriormente, sabemos que necesitamos tres filtros, y cada filtro tiene Parámetros  $3 \times 3 \times 2$ , más un término de sesgo.

Nuestra pila de entrada es

$$\begin{array}{r}
 & 2 & 1 & -1 & 0 & & 3 \\
 & 3 & 0 & 2 & & 2 & 0 \\
 0: & -1 & 2 & -1 & 3 & & 1 \\
 & 3 & 1 & 3 & -1 & -2 & \\
 & 2 & 1 & 0 & 0 & & 3
 \end{array}$$
  

$$\begin{array}{r}
 & & & & 0-2 & 3 \\
 & & & & 1 & 2 \\
 & & & & 1 & 1 \\
 1: & -2 & -3 & -3 & 0 & 1 & \\
 & 0 & 0 & 1 & & 0 & 1 \\
 & -1 & -1 & 0 & -1 & 2 &
 \end{array}$$

Hemos dividido la tercera dimensión para mostrar los dos canales de entrada, cada uno de  $5 \times 5$ .

Los tres filtros son

$$\begin{array}{c}
 \begin{array}{c} f_0 \\ -1 0 -1 \\ 1 1 \\ 2 2 \end{array} & \begin{array}{c} f_1 \\ 2 \\ -1 2 1 \\ -2 -2 0 \end{array} & \begin{array}{c} f_2 \\ 2 \\ 3 -1 0 -1 \\ -2 -2 3 \end{array} \\
 0: & & 1 \\
 \begin{array}{c} 1 \\ 0 \\ -1 -1 -1 \end{array} & \begin{array}{c} 1 0 1 \\ 1 1 1 \\ 0 0 1 \end{array} & \begin{array}{c} 1 0 \\ 0 1 \\ 1 0 -1 \end{array} \\
 1: & & 0
 \end{array}$$

Nuevamente, hemos separado la tercera dimensión. Observe cómo cada filtro tiene dos núcleos de  $3 \times 3$ , uno para cada canal de la entrada de  $5 \times 5 \times 2$ .

Trabajemos aplicando el primer filtro,  $f_0$ . Necesitamos convolucionar el primer canal de entrada con el primer núcleo de  $f_0$ :

$$\begin{array}{r}
 2 1 -1 0 3 \\
 3 0 \quad 2^1 2 -2^3 1 \quad 0 \\
 3 1 2 1 \quad \times \quad 1 1 \quad = \quad 4 1 1 8 \\
 3 -1 -2 \quad 2 2 \quad 2 \quad 9 8 1 \\
 0 0 3 \quad \quad \quad 15 0 6
 \end{array}$$

Luego, necesitamos convolucionar el segundo canal de entrada con el segundo núcleo de  $f_0$ :

$$\begin{array}{r}
 0 -2 3 \\
 1 2 \quad 1 1 -1 -1 3 \\
 -2 -3 -3 0 1 \quad \times \quad 1 1 0 \quad 1 \quad = \quad 10 5 \quad 4 \\
 0 \quad 0 \quad 1 \quad 0 1 \quad -1 -1 -1 \quad 1 -2 -1 \\
 -1 -1 0 -1 2 \quad -6 -4 -3
 \end{array}$$

Finalmente, agregamos las dos salidas de convolución junto con el escalar de polarización simple:

$$\begin{array}{r}
 4 1 1 8 \quad 10 \quad 5 4 \quad 14 \quad 16 \quad 11 \quad 15 \quad 17 \quad 12 \\
 f_0 \quad - \quad + \quad 1 -2 -1 \quad = \quad 10 \quad 6 0 \quad + 1 = \quad 11 \quad 7 1 \\
 9 8 1 \quad -6 -4 -3 \quad 9 -4 3 \quad 10 -3 4
 \end{array}$$

Ya tenemos la primera salida  $3 \times 3$ .

Repetiendo el proceso anterior para  $f_1$  y  $f_2$  da

$$\begin{array}{r}
 4 4 6 \quad -1 \quad 2 -9 \\
 f_1 \quad 0 -7 15 \quad . f_2 \quad 11 -13 -2 -16 \\
 -5 6 2 \quad \quad \quad 2 \quad 6
 \end{array}$$

Completamos la capa convolucional y generamos la salida  $3 \times 3 \times 3$ .

Muchos kits de herramientas facilitan agregar operaciones en la llamada que configura la capa convolucional, pero, conceptualmente, estas son capas propias que aceptan la salida de  $3 \times 3 \times 3$  como entrada. Por ejemplo, si se solicita, Keras aplicará un ReLU a la salida. Aplicar un ReLU, una no linealidad, a la salida de la convolución nos daría

$$\begin{array}{r} 15 \ 17 \ 12 \\ 11 \ 7 \ 1 \\ \hline f_0 \ - \ 10 \ 0 \ 4 \end{array} \quad \begin{array}{r} 4 \ 4 \ 6 \\ 0 \ 0 \ 15 \\ \hline f_1 \ 0 \ 6 \ 2 \end{array} \quad \begin{array}{r} 0 \ 2 \ 0 \\ 11 \ 0 \ 0 \\ \hline f_2 \ 0 \ 2 \ 6 \end{array}$$

Tenga en cuenta que todos los elementos menores que cero ahora son cero. Usamos una no linealidad entre capas convolucionales por la misma razón que usamos una función de activación no lineal en una red neuronal tradicional: para evitar que las capas convolucionales colapsen en una sola capa. Observe cómo la operación para generar las salidas del filtro es puramente lineal; cada elemento de salida es una combinación lineal de valores de entrada. Agregar ReLU rompe esta linealidad.

Una de las razones para la creación de capas convolucionales fue reducir la cantidad de parámetros aprendidos. Para el ejemplo anterior, la entrada fue  $5 \times 5 \times 2 = 50$  elementos. El resultado deseado era  $3 \times 3 \times 3 = 27$  elementos. Una capa completamente conectada entre estas necesitaría aprender  $50 \times 27 = 1350$  pesos, más otros 27 valores de sesgo. Sin embargo, la capa convolucional aprendió tres filtros, cada uno con pesos de  $3 \times 3 \times 2$ , así como tres valores de sesgo, para un total de  $3(3 \times 3 \times 2) + 3 = 57$  parámetros. Agregar la capa convolucional ahorró aprender unos 1300 pesos adicionales.

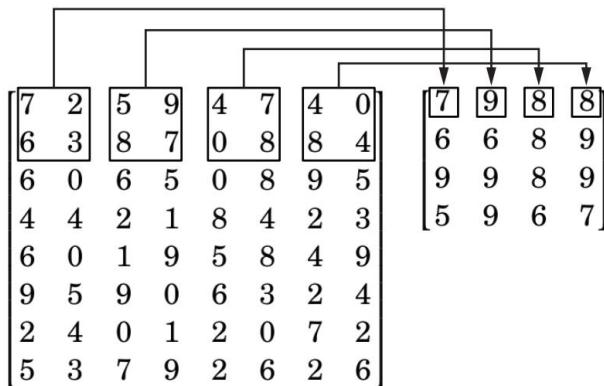
La salida de una capa convolucional suele ser la entrada a una capa de agrupación. Consideremos ese tipo de capa a continuación.

#### Capas de agrupación

Las redes convolucionales a menudo utilizan capas de agrupación después de las capas convolucionales. Su uso es un poco controvertido, ya que descartan información y la pérdida de información podría dificultar que la red aprenda relaciones espaciales. La agrupación generalmente se realiza en el dominio espacial a lo largo de la altura y el ancho del tensor de entrada preservando al mismo tiempo el número de canales.

La operación de agrupación es sencilla: mueve una ventana sobre la imagen, generalmente de  $2 \times 2$  con un paso de dos, para agrupar valores. La operación de agrupación específica realizada en cada grupo es máxima o promedio. La operación de agrupación máxima conserva el valor máximo en la ventana y descarta el resto. La agrupación promedio toma la media de los valores en la ventana.

Una ventana de  $2 \times 2$  con un paso de dos resulta en una reducción de un factor de dos en cada dirección espacial. Por lo tanto, un tensor de entrada (24,24,32) conduce a un tensor de salida (12,12,32). La Figura 9-5 ilustra el proceso para la agrupación máxima.

Figura 9-5: Agrupación máxima con una ventana de  $2 \times 2$  y una zancada de dos

Un canal de entrada, con una altura y una anchura de ocho, está a la izquierda. La ventana de  $2 \times 2$  se desliza sobre la entrada, saltando de dos en dos, por lo que no hay superposición de ventanas. La salida para cada región de  $2 \times 2$  de la entrada es el valor máximo. En cambio, la agrupación promedio generaría la media de los cuatro números. Al igual que con la convolución normal, al final de la fila, la ventana se desliza hacia abajo dos posiciones y el proceso se repite para cambiar el canal de entrada de  $8 \times 8$  a un canal de salida de  $4 \times 4$ .

Como se mencionó anteriormente, la agrupación sin superposición en las ventanas pierde espacio. información especial. Esto ha provocado que algunos miembros de la comunidad de aprendizaje profundo, en particular Geoffrey Hinton, lamenten su uso, ya que dejar caer información espacial distorsiona la relación entre objetos o partes de objetos en la entrada. Por ejemplo, aplicar una ventana de agrupación máxima de  $2 \times 2$  con un paso de uno en lugar de dos a la matriz de entrada de la Figura 9-5 produce

$$\begin{array}{ccccccc}
 & 7 & 8 & 9 & 9 & 8 & 8 \\
 & 6 & 8 & 8 & 7 & 8 & 9 \\
 & & 6 & 6 & 6 & 8 & 8 \\
 & & 6 & 4 & 9 & 9 & 8 \\
 & & 9 & 9 & 9 & 8 & 8 \\
 & & 9 & 9 & 9 & 6 & 6 \\
 & & 5 & 7 & 9 & 9 & 6
 \end{array}$$

Esta es una salida de  $7 \times 7$ , que solo pierde una fila y una columna de la entrada original de  $8 \times 8$ . En este caso, la matriz de entrada se generó aleatoriamente, por lo que deberíamos esperar una operación de agrupación máxima sesgada hacia ochos y nueves: no hay ninguna estructura que capturar. Por supuesto, este no suele ser el caso en una CNN real, ya que lo que deseamos utilizar es la estructura espacial inherente a las entradas.

La agrupación se utiliza comúnmente en el aprendizaje profundo, especialmente para las CNN, por lo que es esencial comprender qué está haciendo una operación de agrupación y ser consciente de sus posibles peligros. Pasemos ahora al extremo de salida de una CNN, normalmente las capas completamente conectadas.

### Capas completamente conectadas

Una capa completamente conectada en una red profunda es, en términos de pesos y datos, idéntica a una capa normal en una red neuronal tradicional. Muchas redes profundas relacionadas con la clasificación pasan la salida de un conjunto de convolución.

y agrupando capas a la primera capa completamente conectada a través de una capa que se aplana el tensor, esencialmente descomponiéndolo en un vector. Una vez que la salida es un vector, la capa completamente conectada utiliza una matriz de peso de la misma manera que lo hace una red neuronal tradicional para asignar una entrada vectorial a una salida vectorial.

### Flujo de datos a través de una red neuronal convolucional

Juntemos todas las piezas para ver cómo fluyen los datos a través de una CNN desde entrada a salida. Usaremos una CNN simple entrenada en el conjunto de datos MNIST, una colección de imágenes en escala de grises de  $28 \times 28$  píxeles de dígitos escritos a mano. La arquitectura se muestra a continuación.

Entrada → Conv(32) → Conv(64) → Pool → Aplanar → Denso(128) → Denso(10)

La entrada es una imagen en escala de grises de  $28 \times 28$  píxeles (un canal). Las capas convolucionales (conv) utilizan núcleos de  $3 \times 3$  y convolución válida, por lo que la altura y el ancho de su salida son dos menos que su entrada. El primer convolucional

La capa aprende 32 filtros mientras que la segunda aprende 64. Estamos ignorando las capas que no afecta la cantidad de datos en la red, como las capas ReLU después las capas convolucionales. Se supone que la capa de agrupación máxima utiliza un tamaño de  $2 \times 2$  ventana con zancadas de dos. La primera capa completamente conectada (densa) tiene 128 nodos, seguido de una capa de salida de 10 nodos, uno para cada dígito, del 0 al 9.

Los tensores que pasan a través de esta red para una sola muestra de entrada son

$(28,28,1) \rightarrow (26,26,32)$	$\rightarrow (24,24,64)$	$\rightarrow (12,12,64) \rightarrow 9216 \rightarrow 128 \rightarrow 10$	
Aporte	conversión	conversión	Piscina
			Aplanar Denso Denso

La capa aplanada desenreda el tensor  $(12,12,64)$  para formar un vector de 9,216 elementos ( $12 \times 12 \times 64 = 9,216$ ). Pasamos los 9216 elementos que genera la capa de diez planos a través de la primera capa densa para generar 128 valores de salida, y el último paso toma el vector de 128 elementos y lo asigna a 10 valores de salida.

Tenga en cuenta que los valores anteriores se refieren a los datos pasados a través de la red para cada muestra de entrada, una de las N muestras del minibatch. Este no es el igual que los parámetros numéricos (ponderaciones y sesgos) que la red necesitaba para aprender durante el entrenamiento.

La red que se muestra arriba se entrenó en los dígitos MNIST utilizando Keras.

La Figura 9-6 ilustra la acción de la red para dos entradas mostrando, Visualmente, la salida de cada capa. Específicamente, muestra la salida de cada capa para dos imágenes de entrada, que representan un 4 y un 6.

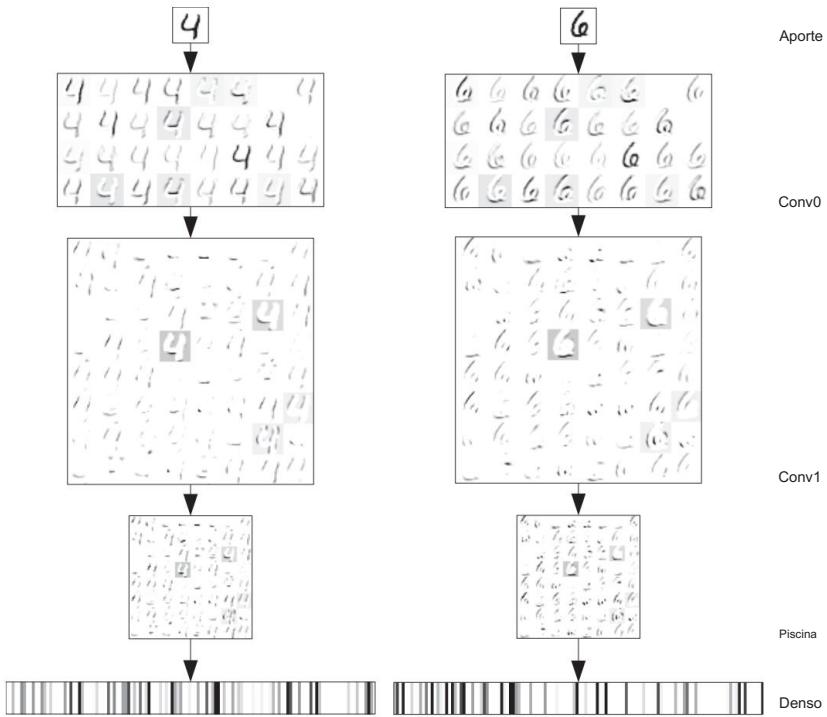


Figura 9-6: Una representación visual de la salida de una CNN para dos entradas de muestra

Comenzando desde arriba, vemos las dos entradas. Para la figura, las intensidades tienen se ha invertido, por lo que más oscuro representa valores numéricos más altos. La entrada es un tensor  $(28, 28, 1)$ , donde el 1 indica una imagen en escala de grises de un solo canal. Una convolución válida con un núcleo de  $3 \times 3$  devuelve una salida de  $26 \times 26$ . La primera capa convolucional aprendió 32 filtros, por lo que la salida es un tensor  $(26, 26, 32)$ . En la figura, mostramos la salida de cada filtro como una imagen. El cero se escala a gris de nivel medio (intensidad 128), los valores más positivos son más oscuros y los valores más negativos son más claros. Vemos diferencias en cómo las entradas se han visto afectadas por los filtros aprendidos. El canal de entrada único significa que cada filtro en esta capa es un único núcleo de  $3 \times 3$ . Las transiciones entre claro y oscuro indican bordes en orientaciones particulares.

Pasamos el tensor  $(26, 26, 32)$  a través de un ReLU (no se muestra aquí) y luego a través de la segunda capa convolucional. La salida de esta capa es un tensor  $(24, 24, 64)$  que se muestra como una cuadrícula de imágenes de  $8 \times 8$  en la figura. Podemos ver muchas partes de los dígitos de entrada resaltadas.

La capa de agrupación conserva el número de canales pero reduce el espacio. dimensión inicial en dos. En forma de imagen, la cuadrícula de  $8 \times 8$  de imágenes de  $24 \times 24$  píxeles es ahora una cuadrícula de  $8 \times 8$  de imágenes de  $12 \times 12$  píxeles. La operación de aplanamiento asigna el tensor  $(12, 12, 64)$  a un vector de 9216 elementos.

La salida de la primera capa densa es un vector de 128 números. En la Figura 9-6, lo mostramos como un código de barras de 128 elementos. Los valores van de izquierda a derecha. La altura de cada barra no es importante y se seleccionó únicamente para que el código de barras sea fácil de ver. El código de barras generado a partir de la entrada.

La imagen es la representación final que utiliza la capa superior de 10 nodos para crear la salida pasada a través de la función softmax. El softmax más alto

La salida se utiliza para seleccionar la etiqueta de clase, "4" o "6".

Por lo tanto, podemos pensar en todas las capas CNN a través de la primera densidad.

capa como mapeo de entradas a una nueva representación, una que facilite un clasificador simple de manejar. De hecho, si pasamos 10 ejemplos de "4" y "6" dígitos a través de esta red y mostrar los vectores de características de 128 nodos resultantes, obtenemos la Figura 9-7, donde podemos ver fácilmente la diferencia entre los patrones de dígitos.

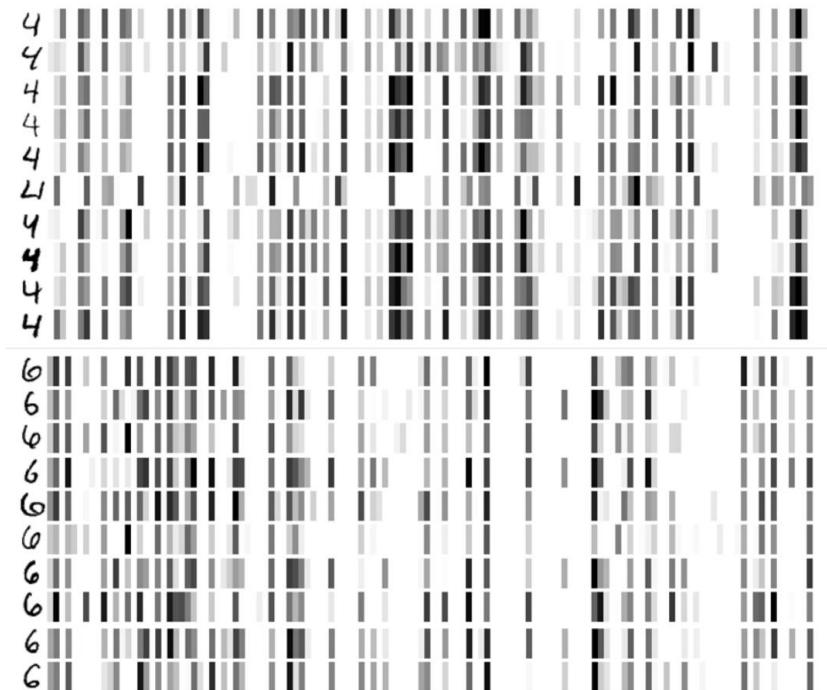


Figura 9-7: Las primeras salidas de capa completamente conectadas para múltiples entradas "4" y "6"

Por supuesto, el objetivo de escribir dígitos como lo hacemos nosotros es hacerlo más fácil. para que los humanos vean las diferencias entre ellos. Si bien podríamos aprender a diferenciar dígitos usando las imágenes vectoriales de 128 elementos, naturalmente preferimos usar los dígitos escritos debido al uso habitual y al hecho de que ya empleamos detectores de características jerárquicas altamente sofisticados a través de nuestro sistema visual del cerebro.

El ejemplo de una CNN aprendiendo una nueva representación de entrada que es más Vale la pena tener en cuenta que es propicio para la interpretación por parte de una máquina, ya que Lo que un humano podría usar en una imagen como pista para su clasificación no es necesariamente lo que una red aprende a usar. Esto podría explicar, en parte, por qué ciertos pasos de preprocessamiento, como los cambios realizados en las muestras de entrenamiento durante El aumento de datos, son tan efectivos para ayudar a la red a aprender a generalizar, cuando muchas de esas alteraciones nos parecen extrañas.

## Resumen

El objetivo de este capítulo era demostrar cómo las redes neuronales manipulan los datos desde la entrada hasta la salida. Naturalmente, no pudimos cubrir todos los tipos de redes, pero, en general, los principios son los mismos: para las redes neuronales tradicionales, los datos se pasan de capa a capa como un vector, y para las redes profundas, se pasan como un tensor. , típicamente de cuatro dimensiones.

Aprendimos cómo presentar datos a una red, ya sea como un vector de características o una entrada multidimensional. Seguimos esto analizando cómo pasar datos a través de una red neuronal tradicional. Vimos cómo los vectores utilizados como entrada y salida de una capa hicieron que la implementación de una red neuronal tradicional fuera un ejercicio sencillo de multiplicación y suma de vectores de matrices.

A continuación, vimos cómo una red convolucional profunda pasa datos de una capa a otra. Primero aprendimos sobre la operación de convolución y luego sobre los detalles de cómo las capas convolucionales y de agrupación manipulan los datos como tensores: un tensor 3D para cada muestra en el minibatch de entrada. En la parte superior de una CNN destinada a la clasificación hay capas completamente conectadas, que vimos actuar exactamente como lo hacen en una red neuronal tradicional.

Terminamos el capítulo mostrando, visualmente, cómo las imágenes de entrada se movían a través de una CNN para producir una representación de salida, permitiendo a la red etiquetar las entradas correctamente. Discutimos brevemente lo que este proceso podría significar en términos de lo que una red capta durante el entrenamiento y cómo eso podría diferir de lo que un humano ve naturalmente en una imagen.

Ahora estamos en condiciones de analizar la retropropagación, el primero de los dos algoritmos críticos que, junto con el descenso de gradiente, hacen posible el entrenamiento de redes neuronales profundas.

# 10

## RETROPROPAGACIÓN



La retropropagación es actualmente el algoritmo central detrás del aprendizaje profundo. Sin él, no podemos entrenar redes neuronales profundas en un período de tiempo razonable, en todo caso. Por lo tanto,

Los profesionales del aprendizaje profundo deben comprender qué es la retropropagación, qué aporta al proceso de capacitación y cómo implementarla, al menos para redes simples. Para los propósitos de este capítulo, asumiré que no tienes conocimientos sobre propagación hacia atrás.

Comenzaremos el capítulo discutiendo qué es la retropropagación y qué no lo es. Luego trabajaremos con los cálculos para una red trivial. Después de eso, presentaremos una descripción matricial de la propagación hacia atrás adecuada para construir redes neuronales de alimentación directa completamente conectadas. Exploraremos las matemáticas y experimentaremos con una implementación basada en NumPy.

Los kits de herramientas de aprendizaje profundo como TensorFlow no implementan la retropropagación como lo haremos en las dos primeras secciones de este capítulo. En lugar de ello, utilizan gráficos computacionales, que analizaremos a alto nivel para concluir el capítulo.

## ¿Qué es la retropropagación?

En el Capítulo 7 introdujimos la idea del gradiente de una función escalar de un vector. Trabajamos nuevamente con gradientes en el Capítulo 8 y vimos su conexión con la matriz jacobiana. Recuerde que en ese capítulo analizamos cómo entrenar una red neuronal es esencialmente un problema de optimización. Sabemos que entrenar una red neuronal implica una función de pérdida, una función de los pesos y sesgos de la red que nos dice qué tan bien se desempeña la red en el conjunto de entrenamiento. Cuando descendemos el gradiente, lo usaremos para decidir cómo pasar de una parte del panorama de pérdidas a otra para encontrar dónde la red funciona mejor. El objetivo del entrenamiento es minimizar la función de pérdida en el conjunto de entrenamiento.

Ésa es la imagen de alto nivel. Ahora hágámoslo un poco más concreto.

Los gradientes se aplican a funciones que aceptan entradas vectoriales y devuelven un valor escalar. Para una red neuronal, el vector de entrada son los pesos y sesgos, los parámetros que definen el rendimiento de la red una vez que se fija la arquitectura. Simbólicamente, podemos escribir la función de pérdida como  $L(\theta)$ , donde ( $\theta$ ) es un vector de todos los pesos y sesgos de la red. Nuestro objetivo es movernos a través del espacio que define la función de pérdida para encontrar el mínimo, el específico que conduce a la pérdida más pequeña,  $L$ . Hacemos esto usando el gradiente de  $L(\theta)$ . Por lo tanto, para entrenar una red neuronal mediante descenso de gradiente, necesitamos saber cómo cada valor de peso y sesgo contribuye a la función de pérdida; es decir, necesitamos saber  $L/w$ , para algún peso (o sesgo)  $w$ .

La retropropagación es el algoritmo que nos dice cuál es  $L/w$  para cada peso y sesgo de la red. Con las derivadas parciales, podemos aplicar el descenso de gradiente para mejorar el rendimiento de la red en el siguiente paso de los datos de entrenamiento.

Antes de continuar, unas palabras sobre terminología. A menudo escucharás a la gente que aprende automáticamente usar la retropropagación como proxy para todo el proceso de entrenamiento de una red neuronal. Los profesionales experimentados entienden lo que quieren decir, pero las personas nuevas en el aprendizaje automático a veces se sienten un poco confundidas. Para ser explícito, la retropropagación es el algoritmo que encuentra la contribución de cada valor de peso y sesgo al error de la red, los  $L/w$ . El descenso de gradiente es el algoritmo que utiliza  $L/w$  para modificar los pesos y sesgos para mejorar el rendimiento de la red en el conjunto de entrenamiento.

Rumelhart, Hinton y Williams introdujeron la propagación hacia atrás en su Artículo de 1986 "Aprendizaje de representaciones mediante errores de retropropagación". En última instancia, la retropropagación es una aplicación de la regla de la cadena que analizamos en los capítulos 7 y 8. La retropropagación comienza en la salida de la red con la función de pérdida. Se mueve hacia atrás, de ahí el nombre de "propagación hacia atrás", a capas cada vez más bajas de la red, propagando la señal de error para encontrar  $L/w$  para cada peso y sesgo. Tenga en cuenta que los profesionales frecuentemente acortan el nombre a "backprop". Encontrarás ese término a menudo.

Trabajaremos con la propagación hacia atrás mediante el ejemplo en las dos secciones siguientes. Por ahora, lo principal que debemos entender es que es la primera de las dos piezas que necesitamos para entrenar redes neuronales. Proporciona la información requerida por la segunda parte, el descenso de gradiente, el tema del Capítulo 11.

## Propagación hacia atrás a mano

Definamos una red neuronal simple, una que acepte dos valores de entrada, tenga dos nodos en su capa oculta y un único nodo de salida, como se muestra en la Figura 10-1.

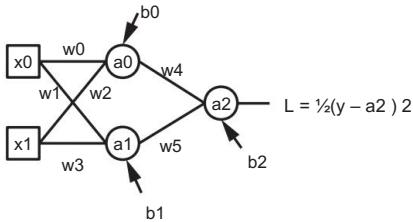


Figura 10-1: Una red neuronal simple

La Figura 10-1 muestra la red con sus seis pesos, de  $w_0$  a  $w_5$ , y tres valores de sesgo,  $b_0$ ,  $b_1$  y  $b_2$ . Cada valor es un escalar.

Usaremos funciones de activación sigmoidea en la capa oculta,

$$(x) = \frac{1}{1 + m_i^{-x}}$$

y sin función de activación para el nodo de salida. Para entrenar la red, usaremos una función de pérdida de error al cuadrado,

$$l = \frac{1}{2} (y - a_2)^2$$

donde  $y$  es la etiqueta, cero o uno, para un ejemplo de entrenamiento y  $a_2$  es la salida de la red para la entrada asociada con  $y$ , es decir,  $x_0$  y  $x_1$ .

Escribamos las ecuaciones para un pase directo con esta red, un pase que se mueve de izquierda a derecha desde la entrada,  $x_0$  y  $x_1$ , hasta la salida,  $a_2$ . Las ecuaciones son

$$z_0 = w_0 x_0 + w_2 x_1 + b_0$$

$$a_0 = (z_0)$$

$$z_1 = w_1 x_0 + w_3 x_1 + b_1$$

$$a_1 = (z_1)$$

$$a_2 = w_4 a_0 + w_5 a_1 + b_2 \quad (10.1)$$

Aquí, hemos introducido valores intermedios  $z_0$  y  $z_1$  como argumentos de las funciones de activación. Observe que  $a_2$  no tiene función de activación. Podríamos haber usado un sigmoide aquí también, pero como nuestras etiquetas son 0 o 1, aprenderemos un buen valor de salida de todos modos.

Si pasamos un único ejemplo de entrenamiento a través de la red, el resultado es Si la etiqueta asociada con el ejemplo de entrenamiento,  $x = (x_0, x_1)$ , es y, el  $a_2$ . La pérdida por error al cuadrado es como se indica en la Figura 10-1.

El argumento de la función de pérdida es  $a_2$ ; y es una constante fija. Sin embargo,  $a_2$  depende directamente de  $w_4, w_5, b_2$  y de los valores de  $a_1$  y  $a_0$ , que a su vez dependen de  $w_0, w_1, w_2, w_3, b_0, b_1, x_0$  y  $x_1$ . Por lo tanto, pensando en términos de ponderaciones y sesgos, podríamos escribir la función de pérdida como

$$L = L(w_0, w_1, w_2, w_3, w_4, w_5, b_0, b_1, b_2; x_0, x_1, y) = L(x, y)$$

Aquí, representa los pesos y sesgos; se considera la variable. Las partes después del punto y coma son constantes en este caso: el vector de entrada  $x = (x_0, x_1)$  y la etiqueta asociada, y.

Necesitamos el gradiente de la función de pérdida,  $L(x, y)$ . Para ser explícitos, nosotros Necesitamos todas las derivadas parciales,  $L/w_5, L/b_0$ , etc., para todas las ponderaciones y sesgos: nueve derivadas parciales en total.

Aquí está nuestro plan de ataque. Primero, trabajaremos con los cálculos para calcular expresiones para las derivadas parciales de los nueve valores. En segundo lugar, escribiremos código Python para implementar las expresiones de modo que podamos entrenar la red de la Figura 10-1 para clasificar las flores de iris. Aprenderemos algunas cosas durante este proceso. Quizás lo más importante es que calcular las derivadas parciales a mano es, dicho sea de paso, tedioso. Lo lograremos, pero veremos en la siguiente sección que, afortunadamente, tenemos una manera mucho más compacta de representar la retropropagación, especialmente para redes feedforward completamente conectadas.

Empecemos.

### Calcular las derivadas parciales

Necesitamos expresiones para todas las derivadas parciales de la función de pérdida de la red de la figura 10-1. También necesitamos una expresión para la derivada de nuestra función de activación, la sigmoidea. Comencemos con el sigmoide, ya que un truco inteligente escribe la derivada en términos del propio sigmoide, un valor calculado durante el pase hacia adelante.

La derivada del sigmoide se muestra a continuación.

$$\begin{aligned}
 f'(x) &= \frac{d}{dx} \left( \frac{1}{1 + mi^{-x}} \right) = (-1 \frac{mi}{(1 + mi^{-x})^2}) (-e^{-x}) \\
 &= \frac{-x}{(1 + mi^{-x})^2} \\
 &= \left( \frac{1}{1 + mi^{-x}} \right) \left( \frac{-x}{mi + mi^{-x}} \right) \\
 &= (x) \left( \frac{-x}{mi + mi^{-x}} \right) \\
 &= (x) \left( \frac{-x - 1}{1 + mi + mi^{-x}} \right) \tag{10.2} \\
 &= (x) \left( \frac{-x}{1 + mi + mi^{-x}} - \frac{1}{1 + mi^{-x}} \right) \\
 &= (x)(1 - (x)) \tag{10.3}
 \end{aligned}$$

El truco de la ecuación 10.2 consiste en sumar y restar uno en el numerador para cambiar la forma del factor para que sea otra copia del propio sigmoide. Entonces, la derivada del sigmoide es el producto del sigmoide por uno menos el sigmoide. Volviendo a la ecuación 10.1, vemos que el pase hacia adelante calcula los sigmoideos, las funciones de activación, como  $a_0$  y  $a_1$ . Por lo tanto, durante la derivación de las derivadas parciales de retropropagación, podremos sustituir  $a_0$  y  $a_1$  mediante la Ecuación 10.3 por la derivada del sigmoide para evitar calcularla una segunda vez.

Empecemos por las derivadas. Fieles al nombre de retropropagación, trabajaremos hacia atrás desde la función de pérdida y aplicaremos la regla de la cadena para llegar a las expresiones que necesitamos. La derivada de la función de pérdida,

$$L = \frac{1}{(y - a_2)^2}$$

es

$$\frac{1}{a_2^2} = (2) \frac{-}{(1 - 2)} (y - a_2)(-1) = a_2 - y \tag{10.4}$$

Esto significa que en todas las expresiones siguientes podemos reemplazar  $L/a_2$  por  $a_2 - y$ . Recuerde que  $y$  es la etiqueta del ejemplo de entrenamiento actual y calculamos  $a_2$  durante el paso directo como salida de la red. Busquemos ahora expresiones para  $w_5$ ,  $w_4$  y  $b_2$ , los parámetros utilizados para calcular  $a_2$ . La regla de la cadena nos dice

$$\frac{I}{w_5} = (\overline{L}\overline{a_2})(\overline{a_2}) = (a_2 - y)a_1 \quad (10.5)$$

desde

$$\frac{a_2}{w_5} = \frac{(w_4a_0 + w_5a_1 + b_2)}{w_5} = a_1$$

Hemos sustituido en la expresión  $a_2$  de la Ecuación 10.1.

Una lógica similar conduce a expresiones para  $w_4$  y  $b_2$ :

$$\begin{aligned} \frac{I}{w_4} &= (\overline{L}\overline{a_2})(\overline{a_2}\overline{w_4}) = (a_2 - y)a_0 \\ \frac{I}{b_2} &= (\overline{L}\overline{a_2})(\overline{a_2}\overline{b_2}) = (a_2 - y)(1) = (a_2 - y) \end{aligned} \quad (10.6)$$

¡Fantástico! Tenemos tres de las derivadas parciales que necesitamos; sólo faltan seis más. Escribamos las expresiones para  $b_1$ ,  $w_1$  y  $w_3$ ,

$$\begin{aligned} \frac{I}{b_1} &= (\overline{L}\overline{a_2})(\overline{a_2}\overline{a_1}\overline{a_1}\overline{z_1})(\overline{b_1}) = (\overline{a_2}\overline{y})w_5a_1(1 - a_1) \\ \frac{I}{w_1} &= (\overline{L}\overline{a_2})(\overline{a_2}\overline{a_1}\overline{a_1}\overline{z_1})(\overline{w_1}) = (\overline{a_2}\overline{y})w_5a_1(1 - a_1)x_0 \\ \frac{I}{w_3} &= (\overline{L}\overline{a_2})(\overline{a_2}\overline{a_1}\overline{a_1}\overline{z_1})(\overline{z_1})(\overline{w_3}) = (a_2 - y)w_5a_1(1 - a_1)x_1 \end{aligned} \quad (10.7)$$

donde usamos

$$\frac{a_1}{z_1} = ' (z_1) = (z_1)(1 - (z_1)) = a_1(1 - a_1)$$

sustituyendo  $a_1$  por  $(z_1)$  mientras calculamos  $a_1$  durante el pase hacia adelante.

Un cálculo similar nos da expresiones para las tres derivadas parciales finales:

$$\frac{\partial}{\partial b_0} = (\text{La}\overline{a_2})(\overline{a_{20}})(\overline{a_{0z}})(\overline{z_0}\overline{\frac{\partial}{\partial b_0}}) = (a_2 - y)w4a_0(1 - a_0)$$

$$\frac{\partial}{\partial w_0} (\text{La}\overline{a_2})(\overline{a_{20}})(\overline{a_{0z}})(\overline{z_0}\overline{\frac{\partial}{\partial w_0}}) = (a_2 - y)w4a_0(1 - a_0)x_0$$

$$\frac{\partial}{\partial w_2} (\text{La}\overline{a_2})(\overline{a_{20}})(\overline{a_{0z}})(\overline{z_0}\overline{\frac{\partial}{\partial w_2}}) = (a_2 - y)w4a_0(1 - a_0)x_1 \quad (10.8)$$

¡Uf! Fue tedioso, pero ahora tenemos lo que necesitamos. Sin embargo, observe que este es un proceso muy rígido: si cambiamos la arquitectura de la red, la función de activación o la función de pérdida, necesitaremos derivar estas expresiones nuevamente. Usemos las expresiones para clasificar las flores de iris.

### Traduciendo a Python El

código que he presentado aquí está en el archivo nn\_by\_hand.py. Échale un vistazo

en un editor para ver la estructura general. Comenzaremos con la función principal

(Listado 10-1):

---

```
épocas = 1000
eta = 0,1
```

```
xtrn, ytrn, xtst, ytst = BuildDataset()
```

```
neto = {}
neto["b2"] = 0,0
neto["b1"] = 0,0
neto["b0"] = 0,0
neto["w5"] = 0,0001*(np.random.random() - 0,5)
neto["w4"] = 0,0001*(np.random.random() - 0,5)
neto["w3"] = 0,0001*(np.random.random() - 0,5)
neto["w2"] = 0,0001*(np.random.random() - 0,5)
neto["w1"] = 0,0001*(np.random.random() - 0,5)
neto["w0"] = 0,0001*(np.random.random() - 0,5)
```

```
tn0,fp0,fn0,tp0,pred0 = Evaluar(net, xtst, ytst)
```

```
neto = GradientDescent(net, xtrn, ytrn, épocas, eta)
```

```
tn,fp,fn,tp,pred = Evaluar(net, xtst, ytst)
```

```
print("Entrenamiento para %d épocas, tasa de aprendizaje %0.5f" % (épocas, eta))
```

```

print()
print("Antes del entrenamiento:")
print("    TN:%3d FP:%3d" % (tn0, fp0))
imprimir("    FN:%3d TP:%3d" % (fn0, tp0))
print()
print("Después del")
entrenamiento:TN:%3d FP:%3d" % (tn, fp))
imprimir("    FN:%3d TP:%3d" % (fn, tp))

```

---

Listado 10-1: La función principal

Primero, establecemos el número de épocas y la tasa de aprendizaje, ( $\eta$ ) . El número de épocas es el número de pasadas por el conjunto de entrenamiento para actualizar los pesos y sesgos de la red. La red es sencilla y nuestro conjunto de datos es pequeño, con solo 70 muestras, por lo que necesitamos muchas épocas para el entrenamiento. El descenso de gradiente utiliza la tasa de aprendizaje para decidir cómo moverse en función de los valores del gradiente. Exploraremos la tasa de aprendizaje más a fondo en el Capítulo 11.

A continuación, cargamos el conjunto de datos . Estamos usando el mismo conjunto de datos de iris que usamos en el Capítulo 6 y nuevamente en el Capítulo 9, manteniendo solo las dos primeras características y clases 0 y 1. Consulte la función BuildDataset en nn\_by\_hand.py. Los valores de retorno son matrices NumPy: xtrn ( $70 \times 2$ ) y xtst ( $30 \times 2$ ) para datos de entrenamiento y prueba, y las etiquetas asociadas en ytrn e ytst.

Necesitamos un lugar para almacenar los pesos y sesgos de la red. Una pitón El diccionario servirá, así que lo configuramos a continuación con los valores predeterminados . Observe que establecemos los valores de sesgo en cero y las ponderaciones en pequeños valores aleatorios en  $[-0,00005, +0,00005]$ . Parecen funcionar bastante bien en este caso.

El resto de main evalúa la red inicializada aleatoriamente (Evaluar ) en los datos de prueba, realiza un descenso de gradiente para entrenar el modelo (GradientDescent ) y evalúa los datos de prueba nuevamente para demostrar que el entrenamiento funcionó .

El Listado 10-2 muestra Evaluate y Forward, que Evaluate llama.

---

```

def Evaluar(net, x, y): out =
    Forward(net, x) tn = fp =
    fn = tp = 0 pred = [] for i
    in
    range(len(y)):    c = 0 if
        (out[ i] < 0.5) else 1 pred.append(c) si (c
        == 0) y (y[i] ==
        0): tn += 1

        elif (c == 0) y (y[i] == 1): fn += 1

        elif (c == 1) y (y[i] == 0): fp += 1 más:

            tp += 1
    retorno tn,fp,fn,tp,pred

```

```

def Adelante(net, x):
    out = np.zeros(x.shape[0])
    para k en el rango(x.shape[0]):
        z0 = net["w0"]*x[k,0] + neto["w2"]*x[k,1] + neto["b0"] a0 =
            sigmoide(z0) z1 =
                neto["w1"]*x[k,0] + neto["w3"]*x[k,1] + neto["b1"] a1 =
                    sigmoide(z1) out[k]
                    = neto["w4"]*a0 + neto["w5"]*a1 + neto["b2"]
    regresar

```

---

### Listado 10-2: La función Evaluar

Comencemos con Forward, que realiza un paso hacia adelante sobre los datos en x. Despu s de crear un lugar para contener la salida de la red (out), cada entrada se ejecuta a trav s de la red utilizando el valor actual de los par metros .

Observe que el c digo es una implementaci n directa de la Ecuaci n 10.1, sin out[k] en lugar de a2 . Cuando se han procesado todas las entradas, devolvemos las salidas recopiladas a la persona que llama.

Ahora veamos Evaluar. Sus argumentos son un conjunto de caracter sticas de entrada, x, etiquetas asociadas, y, y los par metros de red, net. Evaluar primero ejecuta los datos a trav s de la red llamando a Forward para completarlos . Estas son las salidas sin procesar de punto flotante de la red. Para compararlas con las etiquetas reales, aplicamos un umbral para llamar a las salidas < 0,5 clase 0 y a las salidas  $\geq 0,5$  clase 1. La etiqueta prevista se a ade a pred y se cuenta compar ndola con la etiqueta real en y.

Si las etiquetas real y predicha son ambas cero, el modelo ha identificado un verdadero negativo (TN), una instancia verdadera de clase 0. Si la red predice la clase 0, pero la etiqueta real es clase 1, tenemos un falso negativo (FN), una instancia de clase 1 etiquetada como clase 0. Por el contrario , etiquetar una instancia de clase 0 como clase 1 es un falso positivo (FP). La \'unica opci n que queda es una instancia real de clase 1 etiquetada como clase 1, un verdadero positivo (TP). Finalmente, devolvemos los recuentos y predicciones a la persona que llama.

El Listado 10-3 presenta GradientDescent, que el Listado 10-1 llama . Esto es donde implementamos las derivadas parciales calculadas anteriormente.

```

def GradientDescent(net, x, y, epochs, eta):    para
    e en rango( pocas): dw0 =
        dw1 = dw2 = dw3 = dw4 = dw5 = db0 = db1 = db2 = 0.0

        para k en el rango(len(y)):
            z0 = neto["w0"]*x[k,0] + neto["w2"]*x[k,1] + neto["b0"] a0 =
                sigmoide(z0) z1 =
                    neto["w1"]*x[k,0] + neto["w3"]*x[k,1] + neto["b1"] a1 =
                        sigmoide(z1) a2 =
                            neto["w4"]*a0 + neto["w5"]*a1 + neto["b2"]

            db2 += a2 - y[k]
            dw4 += (a2 - y[k]) * a0

```

```

dw5 += (a2 - y[k]) * a1 db1
+= (a2 - y[k]) * neto["w5"] * a1 * (1 - a1) dw1 += (a2 -
y[k]) * neto["w5"] * a1 * (1 - a1) * x[k,0] dw3 += (a2 - y[k]) *
neto["w5"] * a1 * (1 - a1) * x [k,1] db0 += (a2 - y[k]) * neto["w4"] *
a0 * (1 - a0) dw0 += (a2 - y[k]) * neto["w4"] * a0 * (1 -
a0) * x[k,0] dw2 += (a2 - y[k]) * neto["w4"] * a0 * (1 - a0) * x[k,1]

```

```

m = len(y)
neto["b2"] = neto["b2"] - eta * db2 / m
neto["w4"] = neto["w4"] - eta * dw4 / m
neto["w5"] = neto["w5"] - eta * dw5 / m
neto["b1"] = neto["b1"] - eta * db1 / m
neto["w1"] = neto["w1"] - eta * dw1 / m
neto["w3"] = neto["w3"] - eta * dw3 / m
neto["b0"] = neto["b0"] - eta * db0 / m
neto["w0"] = neto["w0"] - eta * dw0 / m
neto["w2"] = neto["w2"] - eta * dw2 / m

```

retorno neto

---

#### Listado 10-3: Usando GradientDescent para entrenar la red

La función GradientDescent contiene un bucle doble. El bucle externo abarca épocas, el número de pasos completos a través del conjunto de entrenamiento. El bucle interno abarca los ejemplos de entrenamiento, uno a la vez. El pase hacia adelante es lo primero para calcular la salida,  $a_2$  y los valores intermedios.

El siguiente bloque de código implementa el paso hacia atrás utilizando las derivadas parciales, ecuaciones 10.4 a 10.8, para mover el error (pérdida) hacia atrás a través de la red. Usamos la pérdida promedio sobre el conjunto de entrenamiento para actualizar los pesos y sesgos. Por lo tanto, acumulamos la contribución a la pérdida para cada peso y valor de sesgo para cada ejemplo de entrenamiento. Esto explica cómo agregar cada nueva contribución al total del conjunto de entrenamiento.

Después de pasar cada ejemplo de entrenamiento por la red y acumular su contribución a la pérdida, actualizamos los pesos y sesgos. Las derivadas parciales nos dan el gradiente, la dirección del cambio máximo; sin embargo, queremos minimizar, por lo que nos movemos en la dirección opuesta al gradiente, restando el promedio de la pérdida debido a cada peso y sesgo de su valor actual.

Por ejemplo,

---

```
neto["b2"] = neto["b2"] - eta * db2 / m
```

---

es

$$b2 \leftarrow b2 - \left( \frac{1}{m} \sum_{i=0}^{m-1} \frac{x_i}{b2} \right)$$

donde  $\eta = 0,1$  es la tasa de aprendizaje y  $m$  es el número de muestras en el conjunto de entrenamiento. La suma es sobre el parcial de  $b_2$  evaluado para cada muestra de entrada,  $x_i$ , cuyo valor promedio, multiplicado por la tasa de aprendizaje, se utiliza para ajustar  $b_2$  para la siguiente época. Otro nombre que utilizamos frecuentemente para la tasa de aprendizaje es tamaño del paso. Este parámetro controla la rapidez con la que los pesos y sesgos de la red avanzan a través del panorama de pérdidas hacia un valor mínimo.

Nuestra implementación está completa. Ejecutémoslo para ver qué tan bien funciona.

### Entrenamiento y prueba del modelo

Echemos un vistazo a los datos de entrenamiento. Podemos trazar las características, una en cada eje, para ver qué tan fácil podría ser separar las dos clases. El resultado es la Figura 10-2, con la clase 0 como círculos y la clase 1 como cuadrados.

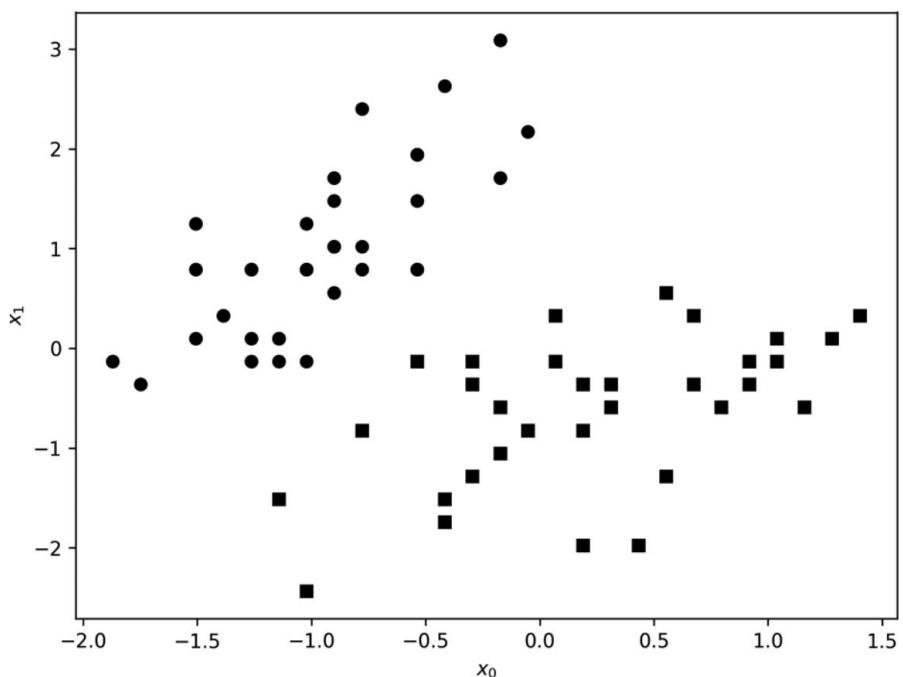


Figura 10-2: Datos de entrenamiento del iris que muestran la clase 0 (círculos) y la clase 1 (cuadrados)

Es sencillo ver que las dos clases están bastante separadas entre sí, por lo que incluso nuestra red elemental con dos neuronas ocultas debería poder aprender la diferencia entre ellas. Compare este gráfico con el lado izquierdo de la Figura 6-2, que muestra las dos primeras características de las tres clases de iris. Si hubiéramos incluido la clase 2 en nuestro conjunto de datos, dos características no serían suficientes para separar las tres clases.

Ejecute el código con

---

`python3 nn_by_hand.py`

---

Para mí esto produce

Entrenamiento para 1000 épocas, tasa de aprendizaje 0.10000

Antes del entrenamiento:

TN: 15 FP: 0

FN: 15 TP: 0

Después de entrenar:

TN: 14 FP: 1

FN: 1 TP: 14

Nos dijeron que el entrenamiento utilizó 1000 pases a través del conjunto de entrenamiento de 70 ejemplos. Este es el bucle exterior del Listado 10-3. Luego se nos presentan dos tablas de números, que caracterizan la red antes y después del entrenamiento. Repasemos estas tablas para comprender la historia que cuentan.

Las tablas se conocen con varios nombres: tablas de contingencia, tablas de  $2 \times 2$  o matrices de confusión. El término matriz de confusión es el más general, aunque suele reservarse para clasificadores multiclase. Las etiquetas cuentan el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos en el conjunto de pruebas. El conjunto de prueba incluye 30 muestras, 15 de cada clase. Si la red es perfecta, todas las muestras de clase 0 estarán en el recuento de TN y todas las de clase 1 en el recuento de TP. Los errores son recuentos de FP o FN.

La red inicializada aleatoriamente etiqueta todo como clase 0. Lo sabemos porque hay 15 muestras TN (aquellas que realmente son clase 0) y 15 muestras FN (15 muestras de clase 1 que están etiquetadas como clase 0). La precisión general antes del entrenamiento es entonces  $15/(15+15) = 0,5 = 50$  por ciento.

Después del entrenamiento, los 1000 pasan por el bucle externo del código en el Listado 10-3, los datos de prueba están casi perfectamente clasificados, con 14 de las 15 etiquetas de clase 0 y 14 de las 15 etiquetas de clase 1 asignadas correctamente. La precisión general ahora es  $(14+14)/(15+15) = 28/30 = 93,3$  por ciento; no está nada mal considerando que nuestro modelo tiene una única capa oculta de dos nodos.

Nuevamente, el objetivo principal de este ejercicio es ver cuán tedioso y potencialmente propenso a errores es calcular las derivadas a mano. El código anterior funciona con escalares; no procesa vectores o matrices para aprovechar cualquier simetría posible mediante el uso de una mejor representación del algoritmo de retropropagación. Afortunadamente, podemos hacerlo mejor. Miremos nuevamente el algoritmo de retropropagación para redes completamente conectadas y veamos si podemos usar vectores y matrices para llegar a un enfoque más elegante.

## Propagación hacia atrás para redes totalmente conectadas

En esta sección, exploraremos las ecuaciones que nos permiten pasar un término de error hacia atrás desde la salida de la red a la entrada. Además, veremos cómo usar este término de error para calcular las derivadas parciales necesarias de los pesos y sesgos de una capa para que podamos implementar el descenso de gradiente. Con todas las expresiones esenciales a mano, implementaremos Python.

clases que nos permitirán construir y entrenar redes neuronales feedforward completamente conectadas de profundidad y forma arbitrarias. Concluiremos probando las clases con el conjunto de datos MNIST.

### Propagando hacia atrás el error

Comencemos con una observación útil: las capas de una red neuronal completamente conectada pueden considerarse funciones vectoriales:

$$y = f(x)$$

donde la entrada a la capa es  $x$  y la salida es  $y$ . La entrada,  $x$ , es la entrada real a la red para una muestra de entrenamiento o, si se trabaja con una de las capas ocultas del modelo, la salida de la capa anterior. Ambos son vectores; cada nodo de una capa produce una única salida escalar que, cuando se agrupa, se convierte en  $y$ , un vector que representa la salida de la capa.

El paso hacia adelante recorre las capas de la red en orden, el mapa se convierte en hacer ping  $x_i$  a  $y$  para  $i = 1 \dots n-1$ , la entrada a la capa  $i + 1$ . Después de todas las capas que se procese  $y$ , usamos la salida de la capa final, llamémosla  $h$ , para calcular la pérdida,  $L(h, y_{\text{verdadero}})$ . La pérdida es una medida de qué tan incorrecta es la red para la entrada,  $x$ , que determinamos comparándolo con la etiqueta verdadera  $y_{\text{verdadero}}$ . Tenga en cuenta que si el modelo es multiclase, la salida  $h$  es un vector, con un elemento para cada clase posible, y la etiqueta verdadera es un vector de ceros, excepto el índice de la etiqueta de clase real, que es uno. Esta es la razón por la que muchos kits de herramientas, como Keras, asignan etiquetas de clases enteras a vectores one-hot.

Necesitamos mover el valor de la pérdida, o el error, nuevamente a través de la red; este es el paso de retropropagación. Para hacer esto para una red completamente conectada utilizando vectores por capa y matrices de peso, primero debemos ver cómo ejecutar el pase directo. Como hicimos con la red que construimos anteriormente, separaremos la aplicación de la función de activación de la acción de una capa completamente conectada.

Por ejemplo, para cualquier capa con el vector de entrada  $x$  proveniente de la capa inferior, necesitamos calcular un vector de salida,  $y$ . Para una capa completamente conectada, el pase hacia adelante es

$$y = Wx + b$$

donde  $W$  es una matriz de peso,  $x$  es el vector de entrada y  $b$  es el vector de sesgo.

Para una capa de activación, tenemos

$$y = g(x)$$

para cualquier función de activación, elegimos. Nos quedaremos con el sigmoide durante el resto de este capítulo. Tenga en cuenta que convertimos la función en una función con valores vectoriales. Para hacer esto, aplicamos la función sigmoidal escalar a cada elemento del vector de entrada para producir el vector de salida:

$$(x) = [ (x_0) (x_1) \dots (x_{n-1}) ]$$

Una red completamente conectada consta de una serie de capas completamente conectadas seguidas de capas de activación. Por lo tanto, el paso hacia adelante es una cadena de operaciones que comienza con la entrada del modelo que se entrega a la primera capa para producir una salida, que luego se pasa a la entrada de la siguiente capa, y así sucesivamente hasta que se hayan procesado todas las capas. .

El pase hacia adelante conduce a la salida final y a la pérdida. La derivada de la función de pérdida con respecto a la salida de la red es el primer término de error. Para devolver el término de error al modelo, necesitamos calcular cómo cambia el término de error con un cambio en la entrada de una capa utilizando cómo cambia el error con un cambio en la salida de la capa. Específicamente, para cada capa, necesitamos saber cómo calcular

$$\frac{mi}{x}$$

Es decir, necesitamos saber cómo cambia el término de error con un cambio en la entrada a la capa dada

$$\frac{mi}{y}$$

que es como cambia el término de error con un cambio en la salida de la capa.  
La regla de la cadena nos dice cómo hacerlo:

$$\frac{mi}{x} = \frac{mi}{y} \frac{y}{x} \quad (10.9)$$

donde  $E/x$  para la capa  $i$  se convierte en  $E/y$  para la capa  $i - 1$  a medida que retrocedemos a través de la red.

Operacionalmente, el algoritmo de retropropagación se convierte en

1. Ejecute un pase hacia adelante para mapear  $x \rightarrow y$ , capa por capa, para obtener el resultado final. salida,  $h$ .
2. Calcule el valor de la derivada de la función de pérdida usando  $h$  y ; esto se convierte en  $E/y$  para la capa de salida.
3. Repita para todas las capas anteriores para calcular  $E/x$  a partir de  $E/y$ , haciendo que  $E/x$  para la capa  $i$  se convierta en  $E/y$  para la capa  $i - 1$ .

Este algoritmo pasa el término de error hacia atrás a través de la red.

Averigüemos cómo obtener las derivadas parciales necesarias por tipo de capa, comenzando con la capa de activación.

Supondremos que conocemos  $E/y$  y buscamos  $E/x$ . La regla de la cadena dice

$$\begin{aligned}
 \frac{\text{mi}}{x} &= \frac{\text{mi}}{y} \frac{y}{x} \\
 &= [\text{mi } \frac{y_0}{x_0} \frac{\text{mi}}{y_1} \frac{y_1}{x_1} \dots] \\
 &= [\frac{y_0}{x_0} '(\text{x}_0) \frac{y_1}{x_1} '(\text{x}_1) \dots] \\
 &= \frac{-\text{mi}}{y} '(\text{X}) \tag{10.10}
 \end{aligned}$$

Aquí presentamos para representar el producto Hadamard. Recuerde que el producto de Hadamard es la multiplicación por elementos de dos vectores o matrices. (Consulte el Capítulo 5 para un repaso).

Ahora sabemos cómo pasar el término de error a través de una capa de activación. La única otra capa que estamos considerando es una capa completamente conectada. Si ampliamos la ecuación 10.9, obtenemos

$$\begin{aligned}
 \frac{\text{mi}}{x} &= \frac{\text{mi}}{y} \frac{y}{x} \\
 &= W \frac{\text{mi}}{y} \tag{10.11}
 \end{aligned}$$

desde

$$\frac{y}{x} = \frac{(\text{Ax} + \text{b})}{x} = W \quad (\text{diseño del denominador})$$

El resultado es  $W.$ , no  $W$ , porque la derivada de una matriz multiplicada por un vector en notación de denominador es la transpuesta de la matriz en lugar de la matriz misma.

Hagamos una pausa para recapitular y pensar en la forma de las ecuaciones 10.10 y 10.11. Estas ecuaciones nos dicen cómo pasar el término de error hacia atrás de una capa a otra. ¿Cuáles son las formas de estos valores? Para la capa de activación, si la entrada tiene  $k$  elementos, entonces la salida también tiene  $k$  elementos. Por lo tanto, la relación en la Ecuación 10.10 debería mapear un vector de elementos  $k$  a otro vector de elementos  $k$ . El término de error,  $E/y$ , es un vector de  $k$  elementos, al igual que la derivada de la función de activación,  $'(x)$ . Finalmente, el producto de Hadamard entre los dos también genera un vector de elemento  $k$ , según sea necesario.

Para la capa completamente conectada, tenemos una entrada de elemento  $m$ ,  $x$ ; un  $n \times m$ -matriz de peso del elemento,  $W$ ; y un vector de salida,  $y$ , de  $n$  elementos. Entonces necesitamos generar un vector de  $m$  elementos,  $E/x$ , a partir del término de error de  $n$  elementos,  $E/y$ . Multiplicar la transpuesta de la matriz de pesos, una matriz de  $m \times n$  elementos, por el término de error da como resultado un vector de  $m$  elementos, ya que  $m \times n$  por  $n \times 1$  es  $m \times 1$ , un vector columna de  $m$  elementos.

#### Cálculo de las derivadas parciales de los pesos y sesgos Las ecuaciones

10.10 y 10.11 nos indican cómo pasar el término de error hacia atrás a través de la red. Sin embargo, el objetivo de la retropropagación es calcular cómo los cambios en los pesos y sesgos afectan el error para que podamos utilizar el descenso de gradiente. Específicamente, para cada capa completamente conectada, necesitamos expresiones para

$$\frac{\partial E}{\partial W} \text{ y } \frac{\partial E}{\partial b}$$

dado

$$\frac{\partial E}{\partial y} \text{ y } \frac{\partial E}{\partial x}$$

Comencemos con  $E/b$ . Aplicando la regla de la cadena una vez más se obtiene

$$\begin{aligned} \frac{\partial E}{\partial b} &= \frac{\partial E}{\partial y} \frac{\partial y}{\partial b} \\ &= \frac{\partial E}{\partial y} \frac{(Ax + b)}{y} \\ &= \frac{\partial E}{\partial y} (0 + 1) y \\ &= \frac{\partial E}{\partial y} y \end{aligned} \tag{10.12}$$

lo que significa que el error debido al término de polarización para una capa completamente conectada es el mismo que el error debido a la salida.

El cálculo de la matriz de pesos es similar:

$$\begin{aligned}
 \frac{m_i}{W} &= \frac{m_i}{y} \frac{y}{W} \\
 &= \frac{m_i}{y} \frac{(Ax + b)}{W} \\
 &= \frac{m_i}{y} (x + 0) \\
 &= \frac{m_i}{y} x
 \end{aligned} \tag{10.13}$$

La ecuación anterior nos dice que el error debido a la matriz de pesos es producto del error de salida y la entrada,  $x$ . La matriz de peso es una matriz de  $n \times m$  elementos, ya que el paso directo se multiplica por el vector de entrada de  $m$  elementos. Por lo tanto, la contribución al error de los pesos,  $E/W$ , también debe ser una matriz  $n \times m$ . Sabemos que  $E/y$  es un vector columna de  $n$  elementos, y el la transpuesta de  $x$  es un vector fila de  $m$  elementos. El producto exterior de los dos es una matriz  $n \times m$ , según sea necesario.

Las ecuaciones 10.10, 10.11, 10.12 y 10.13 se aplican a un único ejemplo de entrenamiento. Esto significa que para una entrada específica a la red, estas ecuaciones, especialmente 10.12 y 10.13, nos dicen la contribución a la pérdida por los sesgos y pesos de cualquier capa para esa muestra de entrada.

Para implementar el descenso de gradiente, necesitamos acumular estos errores, el Términos  $E/W$  y  $E/b$ , sobre las muestras de entrenamiento. Luego usamos el valor promedio de estos errores para actualizar las ponderaciones y sesgos al final de cada época o, como lo implementaremos, minibatch. Como el descenso del gradiente es el tema del Capítulo 11, todo lo que faremos aquí es describir cómo usamos la retropropagación para implementar el descenso de gradiente y dejar los detalles para ese capítulo y el código. implementaremos a continuación.

Sin embargo, en general, para entrenar la red, debemos hacer lo siguiente para cada muestra en el minibatch:

1. Pase la muestra a través de la red para crear la salida.

En el camino, necesitamos almacenar la entrada en cada capa, ya que necesitamos para implementar la retropropagación (es decir, necesitamos  $x$  de Equación 10.13).

2. Calcule el valor de la derivada de la función de pérdida, que para us es el error cuadrático medio, para utilizarlo como primer término de error en la propagación hacia atrás.
3. Recorra las capas de la red en orden inverso, calculando  $E/W$  y  $E/b$  para cada capa completamente conectada. Estos valores son acumulado para cada muestra en el minibatch ( $\Delta W, \Delta b$ ).

Cuando se hayan procesado las muestras del minibatch y se hayan acumulado los errores, es hora de dar un paso de descenso de gradiente. Aquí es donde los pesos y sesgos de cada capa se actualizan mediante

$$\begin{aligned} & \frac{1}{m} \sum_{i=1}^m \text{segundo} \leftarrow \text{segundo} - \eta \Delta b \\ & \frac{1}{m} \sum_{i=1}^m \text{segundo} \leftarrow \text{segundo} - \eta \Delta W \end{aligned} \quad (10.14)$$

siendo  $\Delta W$  y  $\Delta b$  los errores acumulados durante el minibatch y  $m$  el tamaño del minibatch. Los pasos repetidos de descenso de gradiente conducen a un conjunto final de ponderaciones y sesgos: una red entrenada.

Esta sección es bastante matemática. La siguiente sección traduce las matemáticas en código, donde veremos que para todas las matemáticas, el código, debido a NumPy y al diseño orientado a objetos, es bastante compacto y elegante. Si no está seguro de las matemáticas, sospecho que el código será de gran ayuda para aclararle las cosas.

### Una implementación de Python

Nuestra implementación tiene el estilo de kits de herramientas como Keras. Queremos tener la capacidad de crear redes arbitrarias y completamente conectadas, por lo que usaremos clases de Python para cada capa y almacenaremos la arquitectura como una lista de capas. Cada capa mantiene sus pesos y sesgos, junto con la capacidad de realizar un pase hacia adelante, un pase hacia atrás y un paso de descenso en gradiente. Para simplificar, usaremos activaciones sigmoideas y la pérdida al cuadrado.

Necesitamos dos clases: ActivationLayer y FullyConnectedLayer. Una clase de red adicional mantiene las piezas juntas y se encarga de la capacitación. Las clases están en el archivo NN.py. (El código aquí fue modificado del código original por Omar Aflak y se usa con su permiso. Consulte el enlace de GitHub en NN.py. Modifiqué el código para usar mini lotes y admitir pasos de descenso de gradiente distintos de cada muestra).

Repasemos cada una de las tres clases, comenzando con ActivationLayer (ver Listado 10-4). La traducción de los cálculos que hemos hecho a forma de código es bastante elegante, en la mayoría de los casos una sola línea de NumPy.

---

#### clase ActivationLayer:

```
def adelante (self, input_data):
    self.input = input_data
    return sigmoide (input_data)
def back (self, salida_error):
    return sigmoid_prime(self.input) * salida_error
def paso(self, eta):
    return
```

---

Listado 10-4: La clase ActivationLayer

El Listado 10-4 muestra ActivationLayer e incluye solo tres métodos: adelante, atrás y paso a paso. El más simple es el paso. No hace nada, ya que una capa de activación no puede hacer nada durante el descenso del gradiente porque no hay pesos ni valores de sesgo.

El método directo acepta el vector de entrada,  $x$ , lo almacena para su uso posterior y luego calcula el vector de salida,  $y$ , aplicando la función de activación sigmoidea.

El método inverso acepta  $E/y$ , el error de salida de la capa superior. Luego devuelve la Ecuación 10.10 aplicando la derivada del sigmoide (sigmoid\_prime) a la entrada establecida durante el paso hacia adelante, multiplicada por elementos por el error.

Las funciones auxiliares sigmoide y sigmoid\_prime son

---

```
def sigmoide(x):
    devuelve 1.0 / (1.0 + np.exp(-x))
def sigmoid_prime(x):
    devuelve sigmoide(x)*(1.0 - sigmoide(x))
```

---

La clase FullyConnectedLayer es la siguiente. Es más complejo que el Clase ActivationLayer , pero no de manera significativa. Consulte el Listado 10-5.

---

```
clase FullyConnectedLayer:
    def __init__(self, tamaño_entrada, tamaño_salida):
        self.delta_w = np.zeros((tamaño_entrada, tamaño_salida))
        self.delta_b = np.zeros((1,tamaño_salida))
        self.passes = 0
        pesos propios = np.random.rand(tamaño_entrada, tamaño_salida) - 0,5
        self.bias = np.random.rand(1, tamaño_salida) - 0,5

    def adelante(self, input_data):
        self.input = input_data
        return np.dot(self.input, self.weights) + self.bias

    def hacia atrás(self, error_salida):
        error_entrada = np.dot(error_salida, self.weights.T)
        error_pesos = np.dot(self.input.T, error_salida)
        self.delta_w += np.dot(self.input.T , error_salida)
        self.delta_b += error_salida
        self.passes += 1
        retorno error_entrada

    def paso(self, eta):
        self.weights -= eta * self.delta_w / self.passes self.bias -=
            eta * self.delta_b / self.passes
        self.delta_w = np.zeros(self.weights.shape) self.delta_b
        = np.zeros(self.bias.shape) self.passes = 0
```

---

Listado 10-5: La clase FullyConnectedLayer

Le decimos al constructor el número de nodos de entrada y salida. El número de nodos de entrada (`input_size`) especifica el número de elementos del vector que entran en la capa. Del mismo modo, `tamaño_salida` especifica el número de elementos en el vector de salida.

Las capas completamente conectadas acumulan errores de peso y sesgo en el mini lote, los términos  $E/W$  en `delta_w` y los términos  $E/b$  en `delta_b`. Cada muestra procesada se cuenta en pasadas.

Debemos inicializar las redes neuronales con valores de peso y sesgo aleatorios; por lo tanto, el constructor establece una matriz de ponderación inicial y un vector de sesgo utilizando valores aleatorios uniformes en el rango  $[-0.5, 0.5]$ . Observe que el vector de sesgo es  $1 \times n$ , un vector de fila. El código invierte el orden de las ecuaciones anteriores para que coincida con la forma en que normalmente se almacenan las muestras de entrenamiento: una matriz en la que cada fila es una muestra y cada columna una característica. El cálculo produce los mismos resultados porque la multiplicación escalar es commutativa: ab

El método hacia adelante guarda el vector de entrada para su uso posterior por parte de los usuarios hacia atrás y hacia atrás. luego calcula la salida de la capa, multiplicando la entrada por la matriz de peso y sumando el término de sesgo .

Sólo quedan dos métodos. El método inverso recibe  $E/y$  (`output_error`) y calcula  $E/x$  (`input_error`),  $E/W$  (`weights_error`) y  $E/b$  (`output_error`). Agregamos los errores al total de errores de ejecución para la capa, `delta_w` y `delta_b`, para el paso a utilizar.

El método de pasos incluye un paso de descenso de gradiente para una conexión completamente conectada. A diferencia del método vacío de `ActivationLayer`, `FullyConnectedLayer` tiene mucho que hacer. Actualizamos la matriz de ponderaciones y el vector de sesgo usando el error promedio, como en la Ecuación 10.14 . Esto implementa el paso de descenso de gradiente sobre el minibatch. Finalmente, reseteamos los acumuladores y el contador para el siguiente minibatch .

La clase `Network` reúne todo, como se muestra en el Listado 10-6.

Red de clase:

```
def __init__(self, detallado=True):
    self.verbose = detallado
    self.capas = []

def agregar(self, capa):
    self.layers.append(capa)

def predecir(self, input_data):
    resultado =
        [] para i en rango(input_data.shape[0]):
            salida = input_data[i] para
            capa en self.layers: salida =
                capa.forward(salida)
            resultado.append( salida)
    devolver resultado

def fit(self, x_train, y_train, minibatches, learning_rate, lote_size=64): para i en el rango
    (minibatches):
```

```

errar = 0
idx = np.argsort(np.random.random(x_train.shape[0]))[:batch_size]
x_batch = x_train[idx]
y_batch = y_train[idx]
para j en el rango(batch_size):
    salida = x_batch[j]
    para capa en self.layers:
        salida = capa.adelante(salida)

    err += mse(y_batch[j], salida)

    error = mse_prime(y_batch[j], salida) para
    la capa invertida (self.layers): error =
        Layer.backward(error)

    para capa en self.layers:
        Layer.step(learning_rate) if
        (self.verbose) y ((i%10) == 0): err /=
            lote_size
        print('minibatch %5d/%d error=%0.9f' % (i, minilotes, err))

```

---

#### Listado 10-6: La clase Network

El constructor de la clase Network es sencillo. Configuramos un indicador detallado para alternar la visualización del error medio en el minibatch durante el entrenamiento. Una formación exitosa debería mostrar que este error disminuye con el tiempo. A medida que se agregan capas a la red, se almacenan en capas, que el constructor inicializa . El método add agrega objetos de capa a la red agregándolos a las capas .

Una vez entrenada la red, el método de predicción genera una salida para cada muestra de entrada en input\_data con un paso hacia adelante a través de las capas de la red. Observe el patrón: la muestra de entrada se asigna a la salida; luego, el bucle sobre capas llama al método de avance de cada capa, pasando a su vez la salida de la capa anterior como entrada a la siguiente; y así sucesivamente a través de toda la red. Cuando finaliza el ciclo, la salida contiene la salida de la capa final, por lo que se agrega al resultado, que se devuelve a la persona que llama .

Formar la red es trabajo de fit . El nombre coincide con el método de entrenamiento estándar de sklearn. Los argumentos son la matriz NumPy de vectores de muestra, uno por fila (x\_train), y sus etiquetas como vectores one-hot (y\_train). A continuación viene el número de minilotes a entrenar. Hablaremos de los minilotes en un momento. También proporcionamos la tasa de aprendizaje (eta) y un tamaño de minibatch opcional, tamaño\_batch.

El método de ajuste utiliza un doble bucle. El primero es sobre el número deseado de minilotes . Como aprendimos anteriormente, un minibatch es un subconjunto del conjunto de entrenamiento completo y una época es un paso completo por el conjunto de entrenamiento. El uso de todo el conjunto de entrenamiento se conoce como entrenamiento por lotes, y el entrenamiento por lotes utiliza épocas. Sin embargo, hay buenas razones para no realizar entrenamiento por lotes, como verá en el Capítulo 11, por lo que se introdujo el concepto de minibatch. Lo típico

Los tamaños de minibatches oscilan entre 16 y 128 muestras a la vez. Las potencias de dos se utilizan a menudo para mejorar las herramientas de aprendizaje profundo basadas en GPU. Para nosotros, no hay diferencia entre un minibatch de 64 o 63 muestras en términos de rendimiento.

Seleccionamos la mayoría de los minibatches como conjuntos secuenciales de datos de entrenamiento para garantizar que se utilicen todos los datos. Aquí, somos un poco vagos y en su lugar seleccionamos subconjuntos aleatorios cada vez que necesitamos un minibatch. Esto simplifica el código y añade un lugar más donde la aleatoriedad puede mostrar su utilidad. Eso es lo que nos brinda `idx`, un orden aleatorio de índices en el conjunto de entrenamiento, manteniendo solo el valor del primer tamaño de lote. Luego usamos `x_batch` e `y_batch` para los pases reales hacia adelante y hacia atrás.

El segundo ciclo se realiza sobre las muestras del minibatch . Las muestras se pasan individualmente a través de las capas de la red, llamando hacia adelante tal como lo hace la predicción . Para fines de visualización, el error cuadrático medio real entre la salida del paso directo y la etiqueta de muestra se acumula para el minibatch .

El paso hacia atrás comienza con el término de error de salida, la derivada de la función de pérdida, `mse_prime` . Luego, el pase continúa hacia atrás a través de las capas de la red, pasando el error de salida de la capa anterior como entrada a la capa inferior, reflejando directamente el proceso de paso hacia adelante.

Una vez que el bucle procesa todas las muestras del minibatch , es hora de dar un paso de descenso de gradiente basado en el error medio que cada capa de la red acumuló sobre las muestras . El argumento del paso sólo necesita la tasa de aprendizaje. El minibatch concluye informando el error promedio si se establece detallado para cada décimo minibatch.

Experimentaremos con este código nuevamente en el Capítulo 11 mientras exploramos el descenso de gradientes. Por ahora, probémoslo con el conjunto de datos MNIST para ver qué tan bien funciona.

### Usando la implementación

Tomemos `NN.py` a dar una vuelta. Lo usaremos para construir un clasificador para el conjunto de datos MNIST, que encontramos por primera vez en el Capítulo 9. El conjunto de datos MNIST original consta de imágenes en escala de grises de  $28 \times 28$  píxeles de dígitos escritos a mano con fondos negros. Es un caballo de batalla de la comunidad de aprendizaje automático.

Cambiaremos el tamaño de las imágenes a  $14 \times 14$  píxeles antes de convertirlas en vectores de 196 elementos ( $= 14 \times 14$ ).

El conjunto de datos incluye 60.000 imágenes de entrenamiento y 10.000 imágenes de prueba. Los vectores se almacenan en matrices NumPy; vea los archivos en el directorio del conjunto de datos. El código para generar el conjunto de datos está en `build_dataset.py`. Si desea ejecutar el código usted mismo, primero deberá instalar Keras y OpenCV para Python. Keras proporciona el conjunto original de imágenes y asigna las etiquetas del conjunto de entrenamiento a vectores one-hot. OpenCV cambia la escala de las imágenes de  $28 \times 28$  a  $14 \times 14$  píxeles.

El código que necesitamos está en `mnist.py` y se muestra en el Listado 10-7.

---

```
importar numpy como np
desde importación NN
```

```

x_train = np.load("dataset/train_images_small.npy") x_test =
np.load("dataset/test_images_small.npy") y_train =
np.load("dataset/train_labels_vector.npy") y_test = np.load("conjunto de datos/test_labels.npy")

x_train = x_train.reshape(x_train.shape[0], 1, 14*14) x_train /= 255
x_test =
x_test.reshape(x_test.shape[0], 1, 14*14) x_test /= 255

net = Network()
net.add(FullyConnectedLayer(14*14, 100))
net.add(ActivationLayer())
net.add(FullyConnectedLayer(100, 50))
net.add(ActivationLayer())
net.add( FullyConnectedLayer(50, 10))
net.add(ActivationLayer())

net.fit(x_train, y_train, minibatches=40000, learning_rate=1.0)

out = net.predict(x_test) cm =
np.zeros((10,10), dtype="uint32") para i en
rango(len(y_test)):
    cm[y_test[i],np.argmax(out [yo])] += 1

print()
print(np.array2string(cm))
print()
print("precisión = %0.7f" % (np.diag(cm).sum() / cm.sum(),))

```

---

Listado 10-7: Clasificación de dígitos MNIST

Observe que importamos NN.py justo después de NumPy. Cargamos las imágenes de entrenamiento, las imágenes de prueba y las etiquetas a continuación . La clase Network espera que cada vector de muestra sea un vector de fila de  $1 \times n$ , por lo que remodelamos los datos de entrenamiento de (60000,196) a (60000,1,196), lo mismo que los datos de prueba . Al mismo tiempo, escalamos los datos de 8 bits de [0, 255] a [0, 1]. Este es un paso de preprocesamiento estándar para datos de imágenes, ya que al hacerlo facilita el aprendizaje de la red.

Lo siguiente que viene es construir el modelo . Primero, creamos una instancia de la clase Network . Luego, agregamos la capa de entrada definiendo un FullyConnectedLayer con 196 entradas y 100 salidas. A esto le sigue una capa de activación sigmoidea. Luego agregamos una segunda capa completamente conectada que asigna las 100 salidas de la primera capa a 50 salidas, junto con una capa de activación. Finalmente, agregamos una última capa completamente conectada que asigna las 50 salidas de la capa anterior a 10, el número de clases, además de agregar su capa de activación. Este enfoque imita conjuntos de herramientas comunes como Keras.

El entrenamiento pasa por llamarse apto . Especificamos 40.000 minilotes utilizando el tamaño de minilote predeterminado de 64 muestras. Establecemos la tasa de aprendizaje en 1,0,

que funciona bien en este caso. El entrenamiento dura unos 17 minutos en mi antiguo sistema Intel i5 Ubuntu. A medida que el modelo se entrena, el error medio sobre el minibatch. Cuando finaliza el entrenamiento, pasamos la prueba de los 10.000. Muestre a través de la red y calcule una matriz de confusión de  $10 \times 10$ . Recuerde que las filas de la matriz de confusión son las etiquetas de clase verdaderas, aquí los dígitos reales del 0 al 9. Las columnas corresponden a las etiquetas previstas, el valor más grande de las 10 salidas para cada muestra de entrada. La matriz Los elementos son los recuentos de la frecuencia con la que la etiqueta verdadera era  $i$  y la etiqueta asignada. La etiqueta era  $j$ . Si el modelo es perfecto, la matriz es puramente diagonal; hay No hay casos en los que la etiqueta verdadera y la etiqueta del modelo no coincidan. La precisión general se imprime al final como la suma diagonal dividida por la suma de la matriz, el total número de muestras de prueba.

Mi ejecución de mnist.py produjo

```
error del minilote 39940/40000 = 0,003941790
error del minilote 39950/40000 = 0,001214253
error del minilote 39960/40000 = 0,000832551
error del minilote 39970/40000 = 0,000998448
error del minilote 39980/40000 = 0,002377286
error del minilote 39990/40000 = 0,000850956
```

```
[[ 965      0      1      1      1      5      2      3      2      0]
 [ 0 1121 [[ 0      3      2      0      1      3      0      5      0]
   6      0 1005      4      2      0      3      7      5      0]
 [[ 4      1      6 981      0      4      0      9      4      5]
   2      0      3      0 953      0      5      3      1 15]
 [ 8 [[[ 0      0 10      0 864 4      5      1      4      4]
   2      1      1      3      936 0      0      3      0]
   2      7 19      2      1      0      989      1      7]
   5      0      4      5      3      5      7      3 939      3]
   5      5      2 10      8      2      1      3      6 967]]]
```

precisión = 0,9720000

La matriz de confusión es fuertemente diagonal y la precisión general es 97,2 por ciento. Este no es un resultado tan malo para un conjunto de herramientas simple como NN.py y una red feedforward totalmente conectada. El mayor error que comete la red hecho fue confundir sietes con doses 19 veces (elemento [7,2] de la matriz). El siguiente error más cercano fue confundir cuatros con nueves 15 veces (elemento [4,9]). Ambos errores tienen sentido: los sietes y los doses a menudo parecen similares, al igual que los cuatro y los nueves.

Comenzamos este capítulo con una red que creamos que incluía dos personas: puts, dos nodos en la capa oculta y una salida. El archivo iris.py implementa el mismo modelo adaptando el conjunto de datos a lo que Network espera. Nosotros No revisaré el código, pero ejecútelo. Cuando lo hago, mejoró un poco. rendimiento en el equipo de prueba: 14 de 15 correctos para la clase 0 y 15 de 15 para la clase 1.

Lamentablemente, los métodos de retropropagación detallados aquí y en la sección anterior no son, en última instancia, lo suficientemente flexibles para el aprendizaje profundo. Los kits de herramientas modernos no utilizan estos enfoques. Exploraremos qué hacen los kits de herramientas de aprendizaje profundo cuando se trata de retropropagación.

## Gráficos computacionales

En informática, un gráfico es una colección de nodos (vértices) y aristas que los conectan. Hemos estado usando gráficos todo el tiempo para representar redes neuronales. En esta sección, usaremos gráficos para representar expresiones.

Considere esta simple expresión:

$$y = mx + b$$

Para evaluar esta expresión, seguimos reglas acordadas con respecto a la precedencia de operadores. Seguir las reglas implica una secuencia de operaciones primitivas que podemos representar como una gráfica, como se muestra en la Figura 10-3.

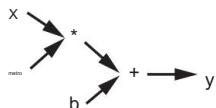


Figura 10-3: Un gráfico computacional que implementa  $y = mx + b$

Los datos fluyen a través del gráfico de la Figura 10-3 a lo largo de las flechas, de izquierda a derecha. Los datos se originan en fuentes, aquí  $x$ ,  $m$ ,  $b$ , y fluyen a través de los operadores,  $*$  y  $+$ , hasta la salida,  $y$ .

La figura 10-3 es un gráfico computacional: un gráfico que especifica cómo evaluar una expresión. Los compiladores de lenguajes como C generan gráficos computacionales de alguna forma para traducir expresiones de alto nivel en secuencias de instrucciones en lenguaje de máquina. Para la expresión anterior, primero se multiplican los valores de  $x$  y  $m$ , y el resultado resultante de la operación de multiplicación se pasa a una operación de suma, junto con  $b$ , para producir el resultado final,  $y$ .

Podemos representar expresiones, incluidas aquellas que representan redes neuronales profundas complejas, como gráficos computacionales. Representamos modelos feedforward completamente conectados de esta manera, como datos que fluyen desde la entrada,  $x$ , a través de las capas ocultas hasta la salida, la función de pérdida.

Los gráficos computacionales son la forma en que los kits de herramientas de aprendizaje profundo como TensorFlow y PyTorch administran la estructura de un modelo e implementan la retropropagación. A diferencia de los cálculos rígidos anteriores en este capítulo, un gráfico computacional es genérico y capaz de representar todas las arquitecturas utilizadas en el aprendizaje profundo.

A medida que lea detenidamente la literatura sobre aprendizaje profundo y comience a trabajar con conjuntos de herramientas específicos, se encontrará con dos enfoques diferentes para utilizar

gráficos computacionales. El primero genera el gráfico dinámicamente cuando hay datos disponibles. PyTorch utiliza este método, llamado símbolo a número. TensorFlow utiliza el segundo método, símbolo a símbolo, para crear un gráfico computacional estático con anticipación. Ambos enfoques implementan gráficos y ambos pueden calcular automáticamente las derivadas necesarias para la retropropagación.

TensorFlow genera los derivados que necesita para la retropropagación de la misma manera que lo hicimos en la sección anterior. Al igual que la suma, cada operación sabe cómo crear la derivada de sus salidas con respecto a sus entradas. Eso, junto con la regla de la cadena, es todo lo que se necesita para implementar la retropropagación. La forma exacta en que se recorre el gráfico depende del motor de evaluación del gráfico y de la arquitectura del modelo específico, pero el gráfico se recorre según sea necesario tanto para el paso hacia adelante como hacia atrás. Tenga en cuenta que debido a que el gráfico computacional divide las expresiones en operaciones más pequeñas, cada una de las cuales sabe cómo procesar gradientes durante el paso hacia atrás (como hicimos anteriormente para ActivationLayer y FullyConnectedLayer), es posible usar funciones personalizadas en capas sin trabajar con las derivadas. . El motor gráfico lo hace por usted, siempre y cuando utilice operaciones primitivas que el motor ya admite.

Repasemos los pasos hacia adelante y hacia atrás de un gráfico computacional. Este ejemplo proviene del artículo de 2015 “TensorFlow: aprendizaje automático a gran escala en sistemas distribuidos heterogéneos” (<https://arxiv.org/pdf/1603.04467.pdf> ).

Una capa oculta en un modelo completamente conectado se expresa como

$$y = (Wx + b)$$

para la matriz de peso W, el vector de polarización b, la entrada x y la salida y.

La figura 10-4 muestra la misma ecuación que un gráfico computacional.

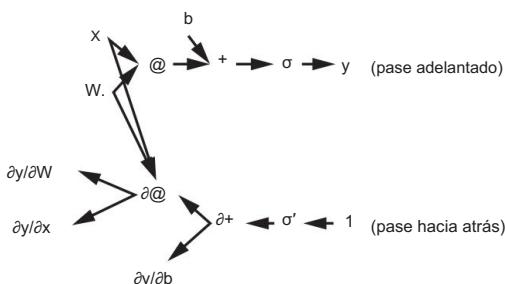


Figura 10-4: Los gráficos computacionales que representan los pasos hacia adelante y hacia atrás a través de una capa de una red neuronal de avance

La Figura 10-4 presenta dos versiones. La parte superior de la figura muestra el paso hacia adelante, donde los datos fluyen desde x, W y b para producir la salida. Observe cómo las flechas van de izquierda a derecha.

Tenga en cuenta que las fuentes son tensores, aquí vectores o matrices. El exterior Los puestos de operaciones también son tensores. Los tensores fluyen a través del gráfico, de ahí el nombre TensorFlow. La figura 10-4 representa la multiplicación de matrices.

como  $@$ , el operador de multiplicación de matrices NumPy. La función de activación es .

Para el paso hacia atrás, la secuencia de derivadas comienza con  $y$  /  $y = 1$  y regresa a través del gráfico desde la salida del operador hasta las entradas. Si hay Si hay más de una entrada, hay más de una derivada de salida. En la práctica, el motor de evaluación de gráficos procesa el conjunto adecuado de operadores en el orden adecuado. Cada operador tiene disponibles sus derivadas de entrada necesarias. cuando es el turno de ese operador para ser procesado.

La figura 10-4 utiliza antes de un operador para indicar las derivadas la opción generador genera. Por ejemplo, el operador de suma (+) produce dos salidas porque hay dos entradas,  $Wx$  y  $b$ . Lo mismo ocurre con la matriz multiplicación (@). La derivada de la función de activación se muestra como Observe que las flechas van desde  $W$  y  $x$  en el pase directo a la derivada.

tiva del operador de multiplicación de matrices en el paso hacia atrás. Ambos  $W$  y  $x$  son necesarios para calcular  $y$  /  $W$  e  $y$  /  $x$ ; consulte la ecuación 10.13 y Ecuación 10.11, respectivamente. No hay una flecha desde  $b$  hasta el operador de multiplicación de matrices porque  $y$  /  $b$  no depende de  $b$ ; consulte la ecuación 10.12. Si una capa estuviera debajo de lo que se muestra en la Figura 10-4, la salida  $y$  /  $x$  de el operador de multiplicación de matrices se convertiría en la entrada para el paso hacia atrás a través de esa capa, y así sucesivamente.

El poder de los gráficos computacionales hace que los kits de herramientas de aprendizaje profundo modernos sean muy generales y admitan casi cualquier tipo y arquitectura de red. sin sobrecargar al usuario con cálculos de gradiente detallados y altamente tediosos. A medida que continúe explorando el aprendizaje profundo, aprecie lo que Los kits de herramientas son posibles con sólo unas pocas líneas de código.

## Resumen

Este capítulo introdujo la retropropagación, una de las dos piezas necesarias para hacer práctico el aprendizaje profundo. Primero, trabajamos calculando el derivados necesarios a mano para una red diminuta y vi lo laborioso que era proceso fue. Sin embargo, pudimos entrenar con éxito la pequeña red.

A continuación, utilizamos nuestros conocimientos de cálculo matricial del Capítulo 8 para encontrar las ecuaciones para redes multicapa totalmente conectadas y creó un sistema simple kit de herramientas en la misma línea que kits de herramientas como Keras. Con el kit de herramientas, entrenamos con éxito un modelo con alta precisión utilizando el conjunto de datos MNIST. Si bien es eficaz y general en términos del número de capas ocultas y sus tamaños, el El kit de herramientas estaba restringido a modelos completamente conectados.

Terminamos el capítulo con una mirada superficial a cómo el aprendizaje profundo moderno kits de herramientas como TensorFlow implementan modelos y automatizan la retropropagación. El gráfico computacional permite combinaciones arbitrarias de operaciones primitivas, cada una de las cuales puede pasar gradientes hacia atrás según sea necesario, con lo que permitiendo las arquitecturas de modelos complejas que encontramos en el aprendizaje profundo.

La segunda mitad del entrenamiento de un modelo profundo es el descenso de gradiente, que pone en funcionamiento los gradientes calculados mediante retropropagación. ahora volvemos nuestra atención de esa manera.



# 11

## DESCENSO DE GRADIENTE



En este capítulo final, reduciremos un poco la velocidad y consideraremos el descenso de gradiente de nuevo. Comenzaremos revisando la idea del descenso de gradiente usando ilustraciones, discutiendo qué es y cómo funciona. A continuación, exploraremos el significado de estocástico en el descenso de gradiente estocástico. El descenso de gradiente es un algoritmo simple que invita a realizar ajustes, por lo que después de explorar el descenso de gradiente estocástico, consideraremos un ajuste útil y de uso común: el impulso. Concluiremos el capítulo analizando algoritmos de descenso de gradiente adaptativos más avanzados, específicamente RMSprop, Adagrad y Adam.

Este es un libro de matemáticas, pero el descenso de gradientes es en gran medida matemática aplicada, por lo que aprenderemos mediante la experimentación. Las ecuaciones son sencillas y las matemáticas que vimos en capítulos anteriores son relevantes como antecedente. Por lo tanto, considere este capítulo como una oportunidad para aplicar lo que hemos aprendido hasta ahora.

## La idea básica

Ya nos hemos encontrado con el descenso de gradiente varias veces. Conocemos la forma de las ecuaciones básicas de actualización del descenso de gradiente a partir de la Ecuación 10.14:

$$W \leftarrow W - \Delta W$$

$$\text{segundo} \leftarrow \text{segundo} - \Delta b \quad (11.1)$$

Aquí,  $\Delta W$  y  $\Delta b$  son errores basados en las derivadas parciales de las ponderaciones y sesgos, respectivamente; ( $\eta$ ) es el tamaño del paso o la tasa de aprendizaje, un valor que utilizamos para ajustar la forma en que nos movemos.

La ecuación 11.1 no es específica del aprendizaje automático. Podemos usar la misma forma para implementar el descenso de gradiente en funciones arbitrarias. Analicemos el descenso de gradiente utilizando ejemplos 1D y 2D para sentar las bases de su funcionamiento. Usaremos una forma no modificada de descenso de gradiente conocida como descenso de gradiente vainilla.

### Descenso de gradiente en una dimensión

Comencemos con una función escalar de  $x$ :

$$f(x) = 6x^2 - 12x + 3 \quad (11.2)$$

La ecuación 11.2 es una parábola orientada hacia arriba. Por tanto, tiene un mínimo.

Encontremos el mínimo analíticamente estableciendo la derivada en cero y resolviendo para  $x$ :

$$\frac{d}{dx} (6x^2 - 12x + 3) = 12x - 12 = 0$$

$$12x = 12$$

$$x = 1$$

El mínimo de la parábola está en  $x = 1$ . Ahora, usemos gradiente descendente para encontrar el mínimo de la Ecuación 11.2. ¿Cómo deberíamos empezar?

Primero, necesitamos escribir la ecuación de actualización adecuada, la forma de la ecuación 11.1 que se aplica en este caso. Necesitamos el gradiente, que para una función 1D  $f'(x) =$  derivada,  $f$  el descenso se convierte en  $12x - 12$ . Con la derivada, el gradiente es simplemente la

$$f' (x) = x - (12x - 12) x \leftarrow x - \quad (11.3)$$

Observa que restamos  $(12x - 12)$ . Por eso el algoritmo se llama descenso de gradiente. Recuerde que el gradiente apunta en la dirección del cambio máximo en el valor de la función. Estamos interesados en minimizar la función, no en maximizarla, por lo que nos movemos en la dirección opuesta al gradiente hacia valores de función más pequeños; por lo tanto, restamos.

La ecuación 11.3 es un paso de descenso de gradiente. Se mueve desde una posición inicial,  $x$ , a una nueva posición basada en el valor de la pendiente en la posición actual. Nuevamente, la tasa de aprendizaje determina qué tan lejos nos movemos.

Ahora que tenemos la ecuación, implementemos el descenso de gradiente. Trazaremos la ecuación 11.2, elegiremos una posición inicial, digamos  $x = -0,9$ , e iteraremos la ecuación 11.3, trazando el valor de la función en cada nueva posición de  $x$ . Si hacemos esto, deberíamos ver una serie de puntos en la función que se acercan cada vez más a la posición mínima en  $x = 1$ . Escribamos algo de código.

Primero, implementamos la Ecuación 11.2 y su derivada:

---

```
def f(x):
    devuelve 6*x**2 - 12*x + 3
def d(x):
    devuelve 12*x - 12
```

---

A continuación, trazamos la función y luego iteramos la Ecuación 11.3, trazando el nuevo par,  $(x, f(x))$ , cada vez:

---

```
importar numpy como
np importar matplotlib.pyplot como plt

x = np.linspace(-1,3,1000)
plt.plot(x,f(x))

x = -0,9
eta = 0,03
para i en el rango(15):
    plt.plot(x, f(x), marcador='o', color='r')
    x = x - eta * d(x)
```

---

Repasemos el código. Después de importar NumPy y Matplotlib, trazamos la Ecuación 11.2. A continuación, establecemos nuestra posición  $x$  inicial y tomamos 15 pasos de descenso de gradiente. Trazamos antes del paso, por lo que vemos la  $x$  inicial pero no trazamos el último paso, lo cual está bien en este caso.

La línea final es clave. Implementa la Ecuación 11.3. Actualizamos la posición  $x$  actual multiplicando el valor de la derivada en  $x$  por  $= 0,03$  como tamaño del paso. El código anterior está en el archivo gd\_1d.py. Si lo ejecutamos, obtenemos la Figura 11-1.

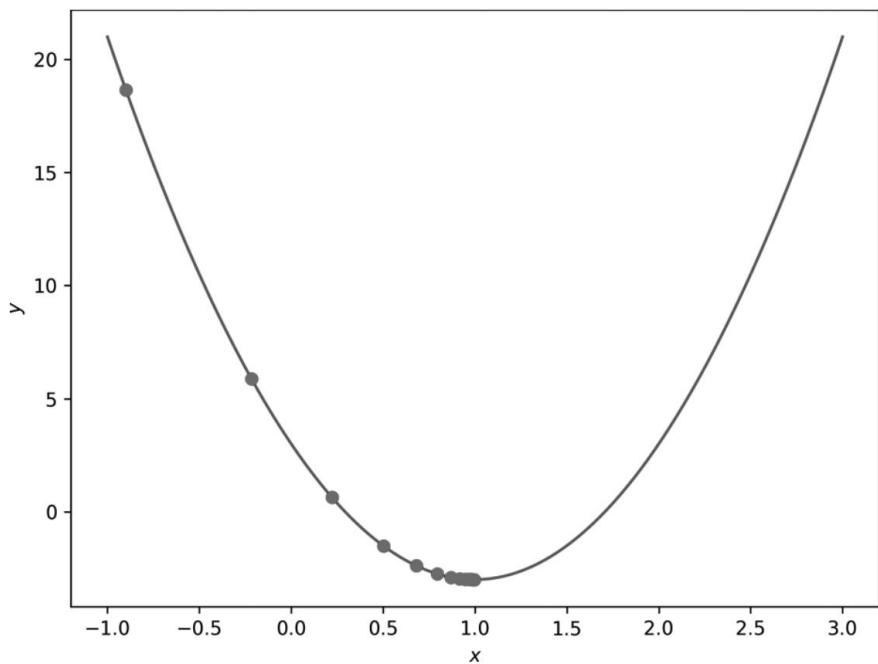


Figura 11-1: Descenso de gradiente en una dimensión con pequeños pasos ( $\alpha = 0,03$ )

Nuestra posición inicial, que podemos considerar como una estimación inicial de la ubicación del mínimo, es  $x = -0,9$ . Claramente, este no es el mínimo. Como nosotros tomamos pasos de descenso de gradiente, nos acercamos sucesivamente al mínimo, a medida que Se muestra la secuencia de círculos que se mueven hacia él.

Note dos cosas aquí. En primer lugar, nos acercamos cada vez más al mínimo. Después de 14 pasos, estamos, para todos los efectos, en el mínimo:

$x = 0,997648$ . En segundo lugar, cada paso de descenso de gradiente conduce a niveles más pequeños y cambios más pequeños en  $x$ . La tasa de aprendizaje es constante en  $= 0,03$ , por lo que la fuente de las actualizaciones más pequeñas de  $x$  deben ser valores cada vez más pequeños de la derivada en cada posición de  $x$ . Esto tiene sentido si lo pensamos. A medida que nos acercamos En la posición mínima, la derivada se hace cada vez más pequeña, hasta que llega a cero en el mínimo, por lo que la actualización utilizando la derivada se vuelve sucesivamente. más pequeño también.

Seleccionamos el tamaño del paso para que la Figura 11-1 se mueva suavemente hacia el mínimo de la parábola. ¿Qué pasa si cambiamos el tamaño del paso? Más a lo largo en gd\_1d.py, el código repite los pasos anteriores, comenzando en  $x = 0,75$  y configurando  $= 0,15$  para realizar pasos que son cinco veces más grandes que los trazados en Figura 11-1. El resultado es la Figura 11-2.

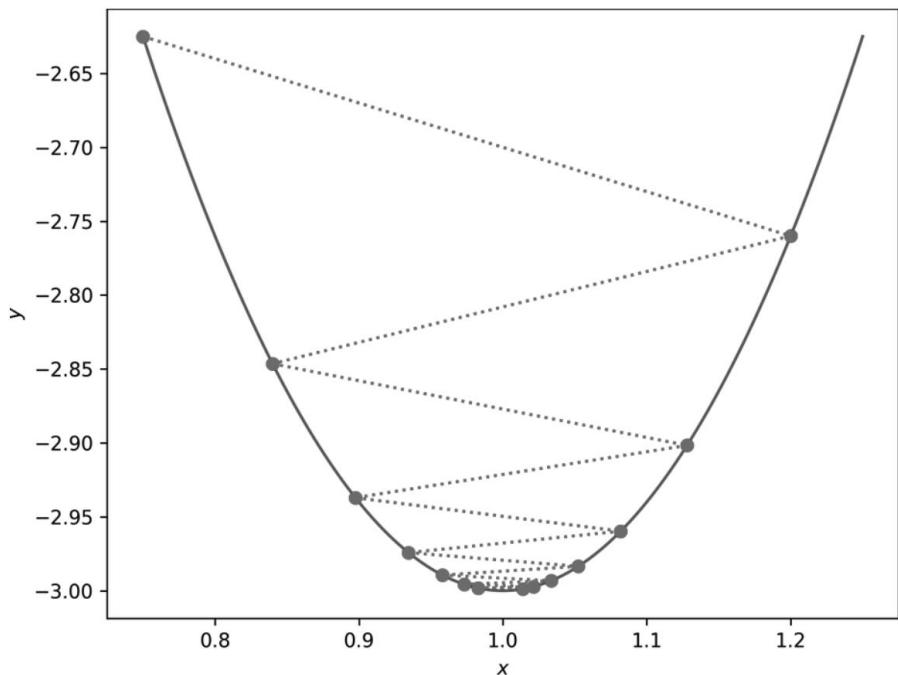


Figura 11-2: Descenso de gradiente en una dimensión con pasos grandes ( $\gamma = 0,15$ )

En este caso, los pasos superan el mínimo. Las nuevas posiciones  $x$  oscilan, rebotando hacia adelante y hacia atrás sobre la posición mínima real. Las líneas discontinuas conectan posiciones  $x$  sucesivas. La búsqueda general todavía se acerca al mínimo, pero tarda más en alcanzarlo, ya que el gran tamaño del paso hace que cada actualización de  $x$  tienda a superar el mínimo.

Los pasos de descenso de gradiente pequeños se mueven distancias cortas a lo largo de la función, mientras que los pasos grandes se mueven distancias grandes. Si la tasa de aprendizaje es demasiado pequeña, se necesitan muchos pasos de descenso de gradiente. Si la tasa de aprendizaje es demasiado grande, la búsqueda se sobrepasa y oscila alrededor de la posición mínima. El ritmo de aprendizaje adecuado no es inmediatamente obvio, por lo que la intuición y la experiencia entran en juego a la hora de seleccionarlo. Además, estos ejemplos se solucionaron. No hay ninguna razón por la que tenga que ser una constante. En muchas aplicaciones de aprendizaje profundo, la tasa de aprendizaje no es constante, sino que evoluciona a medida que avanza el entrenamiento, lo que efectivamente depende del número de pasos de descenso de gradiente realizados.

## Descenso de gradiente en dos dimensiones

El descenso de gradiente en una dimensión es bastante sencillo. Pasemos a dos dimensiones para aumentar nuestra intuición sobre el algoritmo. El código al que se hace referencia a continuación se encuentra en el archivo gd\_2d.py. Primero consideraremos el caso en el que la función tiene un mínimo único y luego veremos casos con múltiples mínimos.

### Descenso de gradiente con un mínimo

único Para trabajar en dos dimensiones, necesitamos una función escalar de un vector,  $f(x) = f(x,y)$ , donde, para que sea más fácil de seguir, separamos el vector en sus componentes. componentes,  $x = (x,y)$ .

La primera función con la que trabajaremos es

$$f(x,y) = 6x^2 + 9y^2 - 12x - 14y + 3$$

Para implementar el descenso de gradiente, también necesitamos las derivadas parciales:

$$\frac{\partial f}{\partial x} = 12x - 12$$

$$\frac{\partial f}{\partial y} = 18y - 14$$

Nuestras ecuaciones de actualización se convierten en

$$x \leftarrow x - \frac{\partial f}{\partial x} = x - (12x - 12)$$

$$y \leftarrow y - \frac{\partial f}{\partial y} = y - (18y - 14)$$

En código, definimos la función y las derivadas parciales:

---

```
def f(x,y):
    devuelve 6*x**2 + 9*y**2 - 12*x - 14*y + 3
def dx(x):
    devuelve 12*x - 12
def dy(y):
    retorno 18*y - 14
```

---

Dado que las derivadas parciales son independientes de la otra variable, podemos pasar solo  $x$  o  $y$ . Veremos un ejemplo más adelante en esta sección en el que ese no es el caso.

El descenso de gradiente sigue el mismo patrón que antes: seleccione una posición inicial, esta vez un vector, repita durante una cierta cantidad de pasos y trace la ruta. La función es 2D, por lo que primero la trazamos usando contornos, como se muestra a continuación.

---

```
norte = 100
x,y = np.meshgrid(np.linspace(-1,3,N), np.linspace(-1,3,N)) z = f(x,y)

plt.contourf(x,y,z, 10, cmap="Grises")
plt.contour(x,y,z,10, colores='k', anchos de línea=1)
plt.plot([0,0],[-1,3],color='k',ancho de línea=1)
plt.plot([-1,3],[0,0],color='k',ancho de línea=1)
plt.plot(1,0.7777778,color='k',marcador= '+')
```

---

Este código requiere alguna explicación. Para trazar contornos, necesitamos una representación de la función sobre una cuadricula de pares ( $x, y$ ). Para generar la grilla, usamos NumPy, específicamente `np.meshgrid`. Los argumentos de `np.meshgrid` son los puntos  $x, y$ , aquí proporcionados por `np.linspace`, que a su vez genera un vector de -1 a 3 de  $N = 100$  valores espaciados uniformemente. La función `np.meshgrid` devuelve dos matrices de  $100 \times 100$ . El primero contiene los valores de  $x$  en el rango dado y el segundo contiene los valores de  $y$ . Todos los pares ( $x, y$ ) posibles están representados en el valor de retorno para formar una cuadricula de puntos que cubre la región de -1...3 tanto en  $x$  como en  $y$ . Al pasar estos puntos a la función, se devuelve  $z$ , una matriz de  $100 \times 100$  del valor de la función en cada par ( $x, y$ ).

Podríamos trazar la función en 3D, pero eso es difícil de ver e innecesario en este caso. En su lugar, usaremos los valores de función en  $x$ ,  $y$  y  $z$  para generar gráficas de contorno. Los gráficos de contorno muestran información 3D como una serie de líneas de igual valor  $z$ . Piense en líneas alrededor de una colina en un mapa topográfico, donde cada línea está a la misma altitud. A medida que la colina aumenta, las líneas encierran regiones sucesivamente más pequeñas.

Los gráficos de contorno vienen en dos variedades, ya sea como líneas de igual valor de función o como sombreados en rangos de la función. Trazaremos ambas variedades usando un mapa en escala de grises. Ese es el resultado neto de llamar a las funciones `plt.contourf` y `plt.contour` de Matplotlib . Las llamadas restantes a `plt.plot` muestran los ejes y marcan el mínimo de la función con un signo más. Los gráficos de contorno son tales que los tonos más claros implican valores de función más bajos.

Ahora estamos listos para trazar la secuencia de pasos de descenso del gradiente. Trazaremos cada posición en la secuencia y las conectaremos con una línea discontinua para aclarar el camino (consulte el Listado 11-1). En código, eso es

---

```
x = xold = -0,5 y
= yold = 2,9
para i en el rango(12):
    plt.plot([xold,x],[yold,y], marcador='o', linestyle='punteado', color='k') xold = x yold
    = yx = x
    - 0,02 *
    dx(x) y = y - 0,02 * dy(y)
```

---

Listado 11-1: Descenso de gradiente en dos dimensiones

Comenzamos en  $(x, y) = (-0,5, 2,9)$  y tomamos 12 pasos de descenso de gradiente. Para conectar la última posición a la nueva posición usando una línea discontinua, realizamos un seguimiento

tanto la posición actual en  $x$  y como la posición anterior,  $(x_{old}, y_{old})$ .

El paso de descenso de gradiente actualiza  $x$  e  $y$  usando  $\Delta = 0,02$  y llamando a las respectivas funciones derivadas parciales,  $dx$  y  $dy$ .

La Figura 11-3 muestra la ruta de descenso del gradiente que sigue el Listado 11-1 (círculos) junto con otras dos rutas que comienzan en  $(1.5, -0.8)$  (cuadrados) y  $(2.7, 2.3)$  (triángulos).

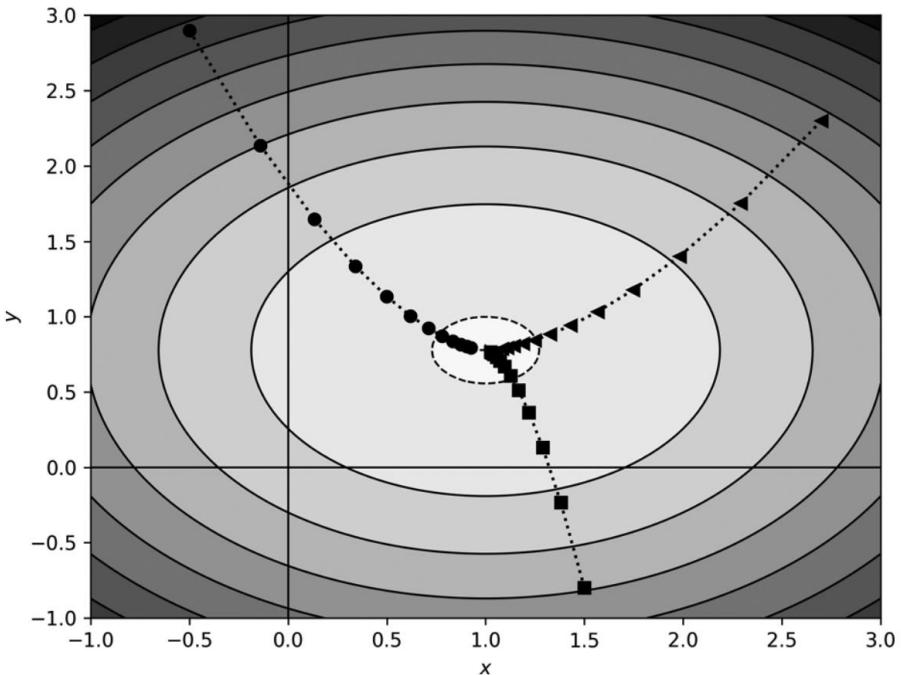


Figura 11-3: Descenso de gradiente en dos dimensiones para pasos pequeños

Las tres rutas de descenso de gradiente convergen hacia el mínimo de la función. Esto no es sorprendente, ya que la función sólo tiene un mínimo. Si la función tiene un mínimo único, entonces el descenso de gradiente eventualmente lo encontrará. Si el tamaño del paso es demasiado pequeño, podrían ser necesarios muchos pasos, pero al final convergerán en el mínimo. Si el tamaño del paso es demasiado grande, el descenso del gradiente puede oscilar alrededor del mínimo pero pasarlo continuamente por encima.

Cambiamos un poco nuestra función para estirarla en la dirección  $x$  en relación con la dirección  $y$ :

$$f(x,y) = 6x^2 + 40y^2 - 12x - 30y + 3$$

Esta función tiene parciales  $f/x = 12x - 12$  y  $f/y = 80y - 30$ .

Además, elegimos dos ubicaciones iniciales,  $(-0.5, 2.3)$  y  $(2.3, 2.3)$ , y generemos una secuencia de pasos de descenso de gradiente con  $\Delta = 0.02$  y  $\Delta = 0.01$ , respectivamente. La Figura 11-4 muestra los caminos resultantes.

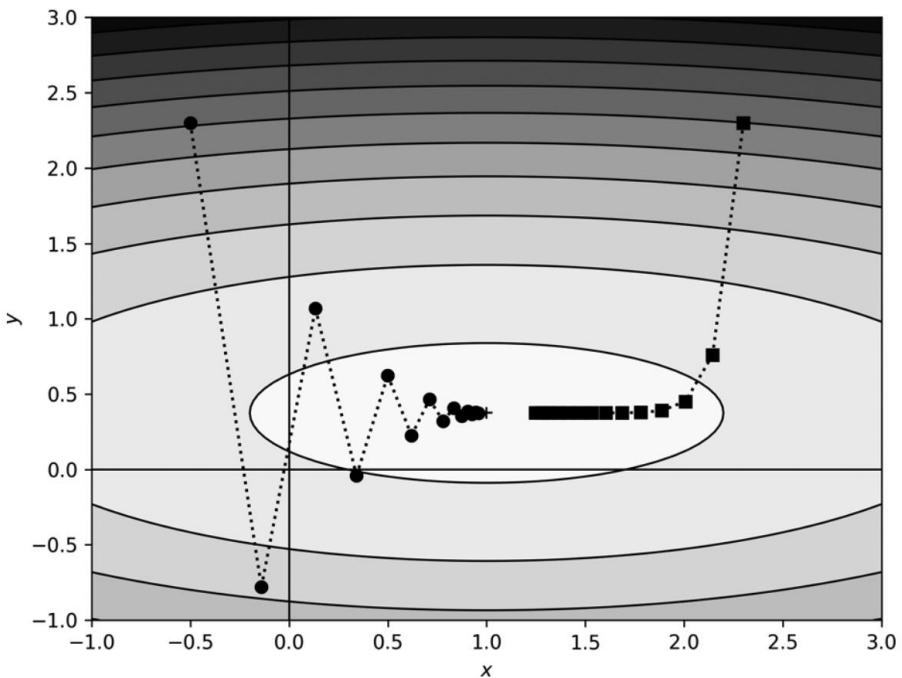


Figura 11-4: Descenso de gradiente en 2D con pasos más grandes y una función ligeramente diferente

Considere primero la ruta = 0,02 (círculo). La nueva función es como un cañón, estrecho en y pero largo en x. El tamaño de paso mayor oscila hacia arriba y hacia abajo en y a medida que se acerca al mínimo en x. Rebotando en las paredes del cañón a un lado, todavía encontramos lo mínimo.

Ahora, eche un vistazo a la ruta = 0,01 (cuadrado). Cae rápidamente al cañón y luego se mueve lentamente sobre la región plana a lo largo del fondo del cañón hacia la posición mínima. El componente del gradiente vectorial (los valores de la derivada parcial de x e y) a lo largo de la dirección x es pequeño en el cañón, por lo que el movimiento a lo largo de x es proporcionalmente lento. No hay movimiento en la dirección y: el cañón es empinado y la tasa de aprendizaje relativamente pequeña ya ha localizado el fondo del cañón, donde el gradiente es principalmente a lo largo de x.

¿Cuál es la lección aquí? Una vez más, el tamaño del paso importa. Sin embargo, la forma de la función importa aún más. El mínimo de la función se encuentra en el fondo de un cañón largo y estrecho. La pendiente a lo largo del cañón es pequeña; el suelo del cañón es plano en la dirección x, por lo que el movimiento es lento porque depende del valor del gradiente. Con frecuencia encontramos este efecto en el aprendizaje profundo: si el gradiente es pequeño, el aprendizaje es lento. Ésta es la razón por la que la unidad lineal rectificada ha llegado a dominar el aprendizaje profundo; el gradiente es constante para entradas positivas. Para una tangente sigmoidea o hiperbólica, el gradiente se acerca a cero cuando las entradas están lejos de cero.

### Descenso de gradiente con múltiples

mínimos Las funciones que hemos examinado hasta ahora tienen un único valor mínimo.

¿Qué pasa si ese no es el caso? Veamos qué sucede con el descenso de gradiente cuando la función tiene más de un mínimo. Considere esta función:

$$f(x,y) = -2 \exp(-1/2((x+1)^2 + (y-1)^2)) - \exp(-1/2((x-1)^2 + (y+1)^2)) \quad (11.4)$$

La ecuación 11.4 es la suma de dos gaussianas invertidas, una con un valor mínimo de  $-2$  en  $(-1, 1)$  y la otra con un mínimo de  $-1$  en  $(1, -1)$ . Si el descenso de gradiente va a encontrar el mínimo global, debería encontrarlo en  $(-1, 1)$ . El código para este ejemplo está en `gd_multiple.py`.

Las derivadas parciales son

$$\frac{\partial f}{\partial x} = 2(x+1) \exp(-1/2((x+1)^2 + (y-1)^2)) + (x-1) \exp(-1/2((x-1)^2 + (y+1)^2)) = 2(y-1) \exp(-1/2((x+1)^2 + (y-1)^2)) - 2(y+1) \exp(-1/2((x-1)^2 + (y+1)^2))$$

$$\frac{\partial f}{\partial y} = ((x+1)^2 + (y-1)^2) + (y+1) \exp(-1/2((x-1)^2 + (y+1)^2)) - (y-1) \exp(-1/2((x+1)^2 + (y-1)^2))$$

lo que se traduce en el siguiente código:

---

```
def f(x,y):
    return -2*np.exp(-0.5*((x+1)**2+(y-1)**2)) + \
           -np.exp(-0.5*((x-1)**2+(y+1)**2))

def dx(x,y):
    devuelve 2*(x+1)*np.exp(-0.5*((x+1)**2+(y-1)**2)) +
           \ (x-1)*np.exp(-0.5*((x-1)**2+(y+1)**2))

def dy(x,y):
    return (y+1)*np.exp(-0.5*((x-1)**2+(y+1)**2)) + \
           2*(y-1)*np.exp(-0.5*((x+1)**2+(y-1)**2))
```

---

Observe que, en este caso, las derivadas parciales dependen tanto de  $x$  como de  $y$ .

El código para la parte de descenso de gradiente de `gd_multiple.py` es el mismo que antes. Ejecutemos los casos de la Tabla 11-1.

Tabla 11-1: Inicio diferente  
Posiciones y número de gradiente

Pasos de descenso tomados

Punto de partida	Pasos	Símbolo (-)
(1,5,1,2)	9	círculo
(1,5,-1,8)	9	cuadrado
(0,0)	20	más
(0,7,-0,2)	20	triángulo
(1,5,1,5)	30	asterisco

La columna Símbolo se refiere al símbolo de trazado utilizado en la Figura 11-5. Para todos los casos,  $\gamma = 0,4$ .

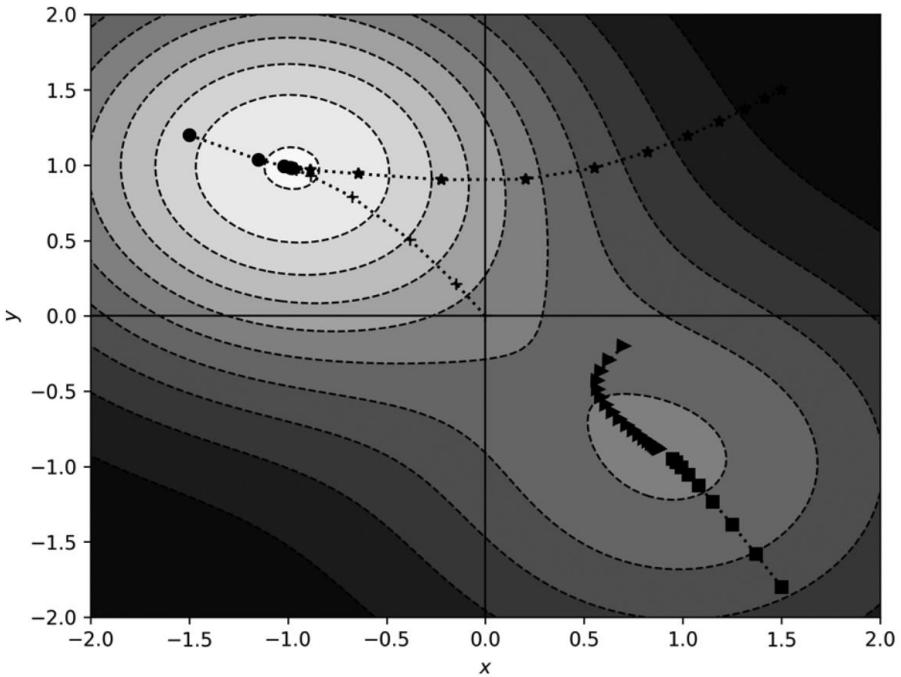


Figura 11-5: Descenso de gradiente para una función con dos mínimos

Las rutas de descenso de gradiente indicadas en la Figura 11-5 tienen sentido. En tres de los cinco casos, el camino se adentra en el pozo que cuento más profundo los dos mínimos definen: una búsqueda exitosa. Sin embargo, para el triángulo y el cuadrado, el descenso del gradiente cayó en el mínimo incorrecto. Claramente, el éxito del descenso de gradiente, en este caso, depende de dónde comenzamos el proceso. Una vez que el camino desciende a una posición más profunda, el descenso en pendiente no tiene forma de escapar hacia arriba para encontrar un mínimo potencialmente mejor.

El pensamiento actual es que el panorama de pérdidas para un modelo de aprendizaje profundo contiene muchos mínimos. Actualmente también se cree que, en la mayoría de los casos, el Los mínimos son bastante similares, lo que explica en parte el éxito de los modelos de aprendizaje profundo: para entrenarlos, no es necesario encontrar el único, mágico y global. mínimo de la pérdida, sólo uno de los (probablemente) muchos que son (probablemente) casi tan bueno como cualquiera de los demás.

Seleccioné las posiciones iniciales utilizadas para los ejemplos de esta sección intencionalmente basándose en el conocimiento de la forma de la función. Para un aprendizaje profundo modelo, elegir el punto de partida significa la inicialización aleatoria de los pesos y prejuicios. En general, no conocemos la forma de la función de pérdida, por lo que la inicialización es un tiro al blanco. La mayor parte del tiempo, o al menos gran parte de Al mismo tiempo, el descenso de gradiente produce un modelo de buen rendimiento. A veces, sin embargo, no es así; fracasa estrepitosamente. En esos casos, es posible que la posición inicial fuera como el cuadrado de la Figura 11-5: cayó en un mínimo local inferior. porque empezó en un mal lugar.

Ahora que conocemos el descenso de gradiente, qué es y cómo se aplica. funciona, investiguemos cómo podemos aplicarlo en el aprendizaje profundo.

## Descenso del gradiente estocástico

Entrenar una red neuronal es principalmente el acto de minimizar la función de pérdida y al mismo tiempo preservar la generalización mediante diversas formas de regularización. En el Capítulo 10, escribimos la pérdida como  $L(\theta; x, y)$  para un vector de pesos y sesgos, ( $\theta$ ), e instancias de entrenamiento  $(x, y)$ , donde  $x$  son los vectores de entrada y  $y$  son las etiquetas conocidas. Observe cómo aquí,  $x$  es un sustituto de todos los datos de entrenamiento, no solo una muestra.

El descenso de gradiente necesita  $L(\theta)$ , que obtenemos mediante retropropagación. El La expresión  $L(\theta)$  es una forma concisa de referirse a todos los pesos individuales. y los términos de error de sesgo que nos proporciona la retropropagación. Obtenemos  $L(\theta)$  promediando el error sobre los datos de entrenamiento. Esto plantea la pregunta: ¿Promediamos más? ¿Todos los datos de entrenamiento o solo algunos de los datos de entrenamiento?

Pasar todos los datos de entrenamiento a través del modelo antes de realizar un paso de descenso de gradiente se denomina entrenamiento por lotes. A primera vista, el entrenamiento por lotes parece sensible. Después de todo, si nuestro conjunto de entrenamiento es una buena muestra de la distribución principal que genera el tipo de datos con el que nuestro modelo pretende trabajar, entonces ¿Por qué no utilizar toda esa muestra para realizar un descenso de gradiente?

Cuando los conjuntos de datos eran pequeños, el entrenamiento por lotes era lo más natural. Sin embargo, los modelos se hicieron más grandes, al igual que los conjuntos de datos, y de repente la carga computacional de pasar todos los datos de entrenamiento a través del modelo para cada uno. El paso de descenso de gradiente se volvió demasiado. Los ejemplos de este capítulo ya sugiere que podrían ser necesarios muchos pasos de descenso de gradiente para encontrar una buena posición mínima, especialmente para tasas de aprendizaje pequeñas.

Por lo tanto, los practicantes comenzaron a utilizar subconjuntos de datos de entrenamiento para cada paso de descenso de gradiente: el minibatch. El entrenamiento en minibatch probablemente fue inicialmente visto como un compromiso, ya que el gradiente calculado sobre el mini lote era "incorrecto" porque no se basaba en el rendimiento del paquete completo.

conjunto de entrenamiento.  
Por supuesto, la diferencia entre lote y minibatch es sólo una ficción acordada. En verdad, es una continuidad desde un minilote de una muestra hasta un minilote de todas las muestras disponibles. Teniendo esto en cuenta, todos los gradientes calculados durante el entrenamiento de la red son "incorrectos" o al menos incompletos, ya que son basados en un conocimiento incompleto del generador de datos y del conjunto completo de datos que podría generar.

Entonces, más que una concesión, la capacitación en minibatch es razonable. El gradiente en un minibatch pequeño es ruidoso en comparación con el calculado en un minibatch más grande, en el sentido de que el gradiente del minibatch pequeño es más grueso estimación del gradiente "real". Cuando las cosas son ruidosas o aleatorias, la palabra el estocástico tiende a aparecer, como ocurre aquí. El descenso de gradiente con minilotes es un descenso de gradiente estocástico (SGD).

En la práctica, el descenso de gradiente utilizando minilotes más pequeños a menudo conduce a Modelos que funcionan mejor que aquellos entrenados con minibatches más grandes. El La razón que se da generalmente es que el gradiente ruidoso del minibatch más pequeño ayuda al descenso del gradiente a evitar caer en mínimos locales pobres del paisaje de pérdidas. Vimos este efecto en la Figura 11-5, donde el triángulo y el cuadrado ambos cayeron en el mínimo equivocado.

Una vez más, nos sentimos extrañamente afortunados. Antes, tuvimos suerte porque el descenso de gradiente de primer orden logró entrenar modelos que no deberíamos entrenar debido a paisajes de pérdidas no lineales, y ahora recibimos un impulso al utilizar intencionalmente pequeñas cantidades de datos para estimar gradientes, omitiendo así una carga computacional que probablemente haría que toda la empresa de análisis profundo El aprendizaje es demasiado complicado de implementar en muchos casos.

¿Qué tamaño debe tener nuestro minibatch? El tamaño del minibatch es un hiperparámetro, algo que debemos seleccionar para entrenar el modelo, pero no forma parte del modelo en sí. El tamaño de minibatch adecuado depende del conjunto de datos. Por ejemplo, en casos extremos, podríamos tomar un paso de descenso de gradiente para cada muestra, que a veces funciona bien. Este caso suele denominarse aprendizaje en línea. Sin embargo, especialmente si usamos capas como la normalización por lotes, necesitamos un mini lote lo suficientemente grande como para calcular las medias y las desviaciones estándar. estimaciones razonables. Nuevamente, como ocurre con casi todo lo demás en el aprendizaje profundo En la actualidad, es empírico y es necesario tener intuición y probar muchas cosas. variaciones para optimizar el entrenamiento del modelo. Por eso la gente trabaja Sistemas AutoML, sistemas que buscan hacer todos los ajustes de hiperparámetros para tú.

Otra buena pregunta: ¿Qué debería haber en el minibatch? Eso es lo que ¿Deberíamos utilizar un pequeño subconjunto del conjunto de datos completo? Normalmente, el orden de la Las muestras en el conjunto de entrenamiento son aleatorias y los minilotes se extraen de el conjunto como fragmentos sucesivos de muestras hasta que se hayan utilizado todas las muestras. El uso de todas las muestras en el conjunto de datos define una época, por lo que el número de muestras en el conjunto de entrenamiento dividido por el tamaño del minibatch determina el número. de minilotes por época.

Alternativamente, como hicimos con NN.py, un minibatch podría ser realmente un muestreo aleatorio de los datos disponibles. Es posible que un entrenamiento particular una muestra nunca se usa mientras que otra se usa muchas veces, pero en general, la mayor parte del conjunto de datos se utiliza durante el entrenamiento.

Algunos kits de herramientas se entran para un número específico de minilotes. Ambos NN.py y Caffe operan de esta manera. Otros kits de herramientas, como Keras y sklearn, utilizan ep-ochs. Los pasos de descenso de gradiente ocurren después de que se procesa un minibatch. Más grande Los minilotes dan como resultado menos pasos de descenso de gradiente por época. Para compensar, los profesionales que utilizan kits de herramientas que utilizan épocas deben asegurarse de que El número de pasos de descenso de gradiente aumenta a medida que aumenta el tamaño del minibatch. Los minilotes más grandes requieren más épocas para entrenarse bien.

En resumen, el aprendizaje profundo no utiliza capacitación por lotes completa por al menos las siguientes razones:

1. La carga computacional es demasiado grande para pasar todo el conjunto de entrenamiento a través del modelo para cada paso de descenso de gradiente.
2. El gradiente calculado a partir de la pérdida promedio en un minibatch es una estimación ruidosa pero razonable del gradiente verdadero y, en última instancia, incognoscible.
3. El gradiente ruidoso apunta en una dirección ligeramente equivocada en el panorama de pérdidas, evitando así posiblemente mínimos malos.
4. El entrenamiento con minibatch simplemente funciona mejor en la práctica para muchos conjuntos de datos.

La razón número 4 no debe subestimarse: muchas prácticas de aprendizaje profundo se emplean inicialmente porque simplemente funcionan mejor. Sólo más tarde se justifican mediante la teoría, si es que se justifica.

Como ya implementamos SGD en el Capítulo 10 (ver NN.py), no lo volveremos a implementar aquí, pero en la siguiente sección agregaremos impulso para ver cómo afecta eso al entrenamiento de redes neuronales.

## Impulso

El descenso del gradiente vainilla se basa únicamente en el valor de la derivada parcial multiplicado por la tasa de aprendizaje. Si el panorama de pérdidas tiene muchos mínimos locales, especialmente si son pronunciados, el descenso del gradiente vainilla podría caer en uno de los mínimos y no poder recuperarse. Para compensar, podemos modificar el descenso del gradiente básico para incluir un término de impulso, un término que utiliza una fracción de la actualización del paso anterior. Incluir este impulso en el descenso del gradiente agrega inercia al movimiento del algoritmo a través del panorama de pérdidas, lo que potencialmente permite que el descenso del gradiente supere los mínimos locales incorrectos.

Definamos y luego experimentemos con el impulso usando ejemplos 1D y 2D, como hicimos antes. Despues de eso, actualizaremos nuestro kit de herramientas NN.py para aprovechar el impulso y ver cómo afecta esto a los modelos entrenados en conjuntos de datos más complejos.

### ¿Qué es el impulso?

En física, el impulso de un objeto en movimiento se define como la masa multiplicada por la velocidad,  $p = mv$ . Sin embargo, la velocidad en sí es la primera derivada de la posición,  $v = dx/dt$ , por lo que el impulso es la masa multiplicada por la rapidez con la que cambia la posición del objeto en el tiempo.

Para el descenso de gradiente, la posición es el valor de la función y el tiempo es el argumento de la función. La velocidad, entonces, es la rapidez con la que cambia el valor de la función con un cambio en el argumento,  $f'/x$ . Por lo tanto, podemos pensar en el impulso como un término de velocidad escalada. En física, el factor de escala es la masa.

Para el descenso de gradiente, el factor de escala es ( $\mu$ ), un número entre cero y uno.

Si llamamos  $v$  al gradiente que incluye el término de impulso, entonces el gradiente ecuación de actualización de descenso ent fue

$$x \leftarrow x - \frac{F}{x}$$

se convierte

$$v \leftarrow v - \frac{F}{x}$$

$$x \leftarrow x + v \quad (11.5)$$

para alguna velocidad inicial,  $v = 0$ , y la “masa”, .

Repasemos la Ecuación 11.5 para entender lo que significa. La actualización de dos pasos, primero  $v$  y luego  $x$ , facilita la iteración, ya que sabemos que debemos hacer para el descenso de gradiente. Si sustituimos  $v$  en la ecuación de actualización por  $x$ , obtenemos

$$x \leftarrow x + x - \frac{F}{x}$$

Esto deja en claro que la actualización incluye el paso de gradiente que teníamos anteriormente, pero lo agrega en una fracción del tamaño del paso anterior. Es una fracción porque restringimos a  $[0, 1]$ . Si  $= 0$ , volvemos al descenso del gradiente vainilla.

Podría ser útil pensar en un factor de escala, la fracción de la anterior velocidad para mantenerse junto con el valor de gradiente actual.

El término de impulso tiende a mantenerse en movimiento a través del panorama de pérdidas. dirigiéndose en su dirección anterior. El valor de determina la fuerza de esa tendencia. Los profesionales del aprendizaje profundo suelen utilizar = 0,9, por lo que la mayoría de la dirección de actualización anterior se mantiene en el siguiente paso, y el gradiente actual proporciona un pequeño ajuste. De nuevo, como muchas cosas en lo profundo Al aprender, este número fue elegido empíricamente.

La primera ley del movimiento de Newton establece que un objeto en movimiento permanece en movimiento a menos que actúe sobre él una fuerza externa. Resistencia a un exterior La fuerza está relacionada con la masa del objeto y se llama inercia. Entonces, también podríamos Considerar el término  $v$  como inercia, que podría haber sido un mejor nombre para él.

Independientemente del nombre, ahora que lo tenemos, veamos qué hace los ejemplos 1D y 2D que trabajamos anteriormente usando gradiente vainilla descendencia.

## Impulso en 1D

Modifiquemos los ejemplos 1D y 2D anteriores para usar un término de impulso. Bien Comience con el caso 1D. El código actualizado está en el archivo gd\_1d\_momentum.py y aparece aquí como Listado 11-2.

---

```

importar matplotlib.pyplot como plt

def f(x):
    devuelve 6*x**2 - 12*x + 3 def
d(x):
    devuelve 12*x - 12

m = ['o','s','>','<', '**','+','p','h','P','D'] x = np.linspace( 0,75,1,25,1000)
plt.plot(x,f(x))

x = xold = 0,75 eta =
0,09
mu = 0,8
v = 0,0

para i en rango(10):
    plt.plot([xold,x], [f(xold),f(x)], marcador=m[i], estilo de línea='punteado',
              color='r') xold
    = xv =
        mu*v - eta * d(x)
    x = x + v

para i en el rango(40): v
    = mu*v - eta * d(x)
    x = x + v

plt.plot(x,f(x),marcador='X', color='k')

```

---

#### Listado 11-2: Descenso de gradiente en una dimensión con impulso

El Listado 11-2 es un poco denso, así que analicémoslo. Primero, estamos trazando, por lo que incluimos Matplotlib. A continuación, definimos la función  $f(x)$  y su derivada,  $d(x)$ , como lo hicimos antes. Para configurar el trazado, definimos una colección de marcadores y luego trazamos la función misma. Como antes, comenzamos en  $x = 0,75$  y establecemos el tamaño del paso (eta), el impulso (mu) y la velocidad inicial (v).

Ahora estamos listos para iterar. Usaremos dos bucles de descenso de gradiente. El primero traza cada paso y el segundo continúa el descenso del gradiente para demostrar que finalmente localizamos el mínimo, que marcamos con una 'X'. Para cada paso, calculamos la nueva velocidad imitando la Ecuación 11.5 y luego sumamos la velocidad a la posición actual para obtener la siguiente posición.

La Figura 11-6 muestra el resultado de gd\_1d\_momentum.py.

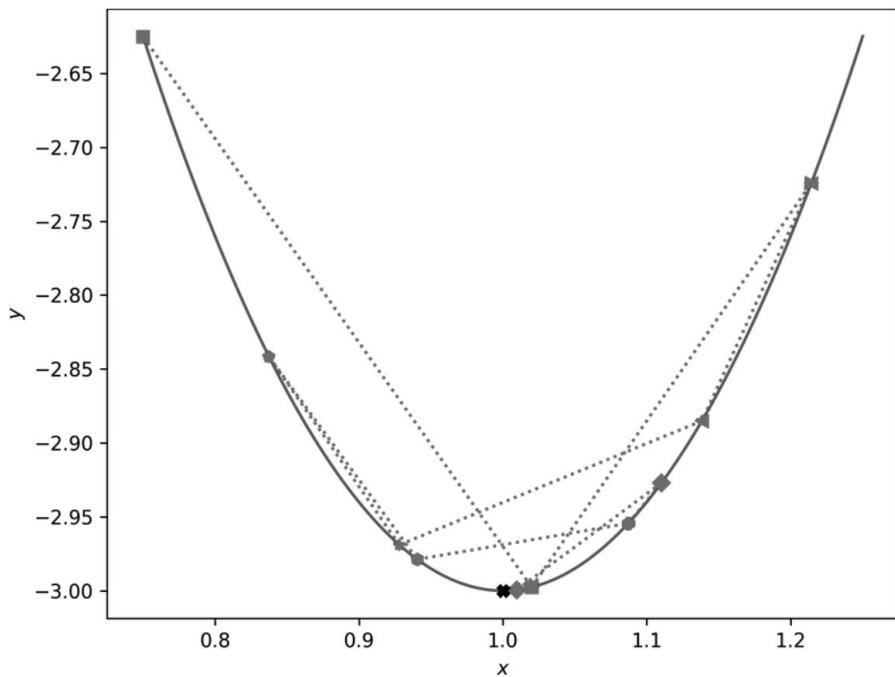


Figura 11-6: Descenso de gradiente en una dimensión con impulso

Tenga en cuenta que utilizamos intencionalmente un tamaño de paso grande ( ), por lo que sobrepasamos el mínimo. El término de impulso también tiende a sobrepasar los mínimos.

Si sigue la línea discontinua y la secuencia de marcadores de trazado, podrá recorrer los primeros 10 pasos de descenso de gradiente. Hay oscilación, pero la oscilación se amortigua y finalmente se estabiliza en el mínimo, como se marca.

Agregar impulso mejoró el exceso debido al gran tamaño del paso. Sin embargo, incluso con el término de impulso, que no es ventajoso aquí, porque solo hay un mínimo, con suficientes pasos de descenso de gradiente, al final encontramos el mínimo.

## Impulso en 2D

Ahora, actualicemos nuestro ejemplo 2D. Estamos trabajando con el código en `gd_momentum.py`. Recuerde que, para el ejemplo 2D, la función es la suma de dos gaussianas invertidas. La inclusión del impulso actualiza ligeramente el código, como se muestra en el Listado 11-3:

---

```
def gd(x,y, eta,mu, pasos, marcador):
    xold = x
    yold = y
    vx = vy = 0.0
    para i en rango(pasos):
        plt.plot([xold,x],[yold,y ], marcador=marcador,
                 estilo de línea='punteado', color='k')
```

```

xold = x
yold = y
vx = mu*vx - eta * dx(x,y) vy =
mu*vy - eta * dy(x,y)   x = x +
vx
y = y + vy

gd( 0.7,-0.2, 0.1, 0.9, 25, '>') gd( 1.5,
1.5, 0.02, 0.9, 90, '**')

```

---

## Listado 11-3: Descenso de gradiente en dos dimensiones con impulso

Aquí tenemos la nueva función, gd, que realiza un descenso de gradiente con impulso que comienza en  $(x,y)$ , utilizando las ejecuciones yy dadas para las iteraciones de pasos .

La velocidad inicial se establece y comienza el ciclo. La actualización de velocidad de La ecuación 11.5 se convierte en  $vx = mu*vx - eta * dx(x,y)$  , y la actualización de posición se convierte en  $x = x + vx$  . Como antes, se traza una línea entre la última posición y la actual para seguir el movimiento a través del paisaje funcional.

El código en gd\_momentum.py rastrea el movimiento comenzando en dos de los puntos que usamos antes,  $(0.7, -0.2)$  y  $(1.5, 1.5)$  . Tenga en cuenta que la cantidad de pasos y la tasa de aprendizaje varían según el punto para evitar que la trama se abarrote demasiado. El resultado de gd\_momentum.py es la Figura 11-7.

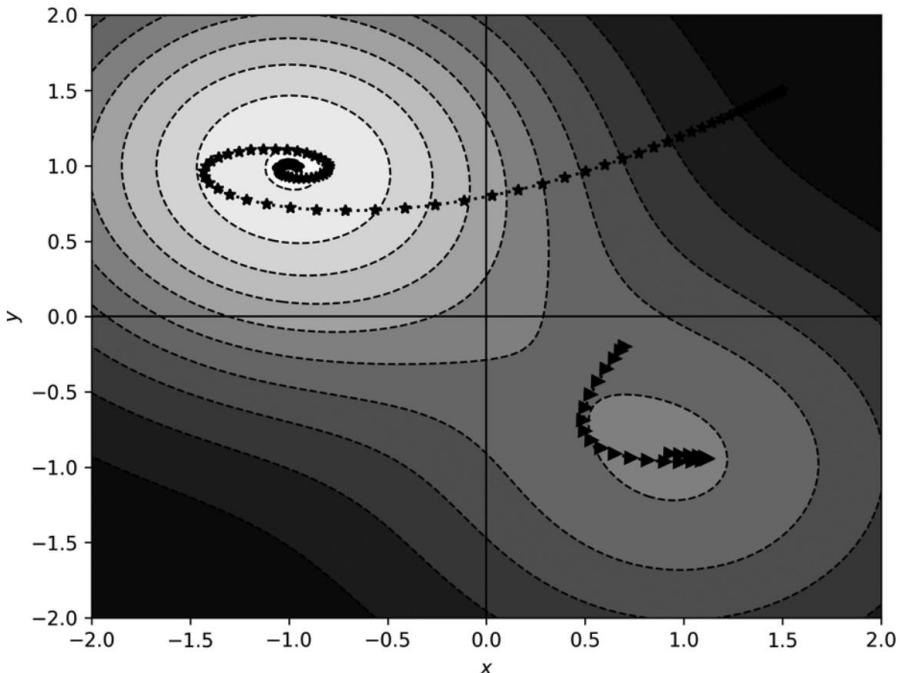


Figura 11-7: Descenso de gradiente en dos dimensiones con impulso

Compare las rutas de la Figura 11-7 con las de la Figura 11-5. Agregar impulso ha empujado los caminos, por lo que tienden a seguir moviéndose en el mismo

dirección. Observe cómo el camino que comienza en (1.5, 1.5) gira en espiral hacia el mínimo, mientras que el otro camino se curva hacia el mínimo menos profundo, lo pasa y retrocede hacia él nuevamente.

El término de momento altera la dinámica del movimiento a través del espacio funcional. Sin embargo, no es inmediatamente evidente que el impulso aporte algo útil. Después de todo, la posición inicial (1.5, 1.5) usando el descenso de gradiente vainilla se movió directamente a la posición mínima sin girar en espiral.

Agreguemos impulso a nuestro kit de herramientas NN.py y veamos si nos ayuda a conseguir algo. al entrenar redes neuronales reales.

#### Modelos de entrenamiento con Momentum

Para respaldar el impulso en NN.py, necesitamos modificar el método FullyConnectedLayer en dos lugares. Primero, como se muestra en el Listado 11-4, modificamos el constructor para permitir una palabra clave de impulso :

---

```
def __init__(self, tamaño_entrada, tamaño_salida, impulso=0.0):
    self.delta_w = np.zeros((tamaño_entrada, tamaño_salida))
    self.delta_b = np.zeros((1,tamaño_salida))
    self.passes = 0
    self.weights = np.random.rand(tamaño_entrada, tamaño_salida) -
    0,5
    self.bias = np.random.rand(1, tamaño_salida) - 0,5
    self.vw = np.zeros((tamaño_entrada, tamaño_salida)) self.vb
    = np.zeros((1, tamaño_salida)) self.momentum
    = impulso
```

---

#### Listado 11-4: Agregar la palabra clave impulso

Aquí, agregamos una palabra clave de impulso , con un valor predeterminado de cero, a la lista de argumentos. Luego, definimos velocidades iniciales para los pesos (vw) y los sesgos (vb) . Estas son matrices de la forma adecuada inicializadas a cero. También mantenemos el argumento del impulso para su uso posterior.

La segunda modificación es al método de pasos , como muestra el Listado 11-5:

---

```
def paso(self, eta):
    self.vw = self.momentum * self.vw - eta * self.delta_w / self.passes
    self.vb = self.momentum * self.vb - eta * self.delta_b / self.passes
    self.weights = self.weights + self.vw
    self.bias = self.bias + self.vb
    self.delta_w = np.zeros(self.weights.shape)
    self.delta_b = np.zeros(self.bias.shape)
    self.passes = 0
```

---

#### Listado 11-5: Actualizando el paso para incluir impulso

Implementamos la Ecuación 11.5, primero para los pesos , luego para los sesgos en la línea siguiente. Multiplicamos el impulso () por la velocidad anterior, luego restamos el error promedio del minibatch, multiplicado por la tasa de aprendizaje. Luego movemos los pesos y los sesgos sumando la velocidad . Eso es todo lo que necesitamos hacer para incorporar impulso. Luego, para utilizarlo, le añadimos el

Palabra clave de impulso para cada capa completamente conectada al construir la red, como se muestra en el Listado 11-6:

---

```
neto = Red()
net.add(FullyConnectedLayer(14*14, 100, impulso=0,9))
net.add(ActivationLayer())
net.add(FullyConnectedLayer(100, 50, impulso=0,9))
net.add(ActivationLayer())
net.add(FullyConnectedLayer(50, 10, impulso=0,9))
net.add(ActivationLayer())
```

---

Listado 11-6: Especificación del impulso al construir la red

Agregar impulso por capa abre la posibilidad de utilizar valores de impulso específicos de cada capa. Si bien no tengo conocimiento de ninguna investigación que lo haga, Parece bastante obvio intentarlo, por lo que a estas alturas es probable que alguien haya experimentado con ello. Para nuestros propósitos, estableceremos el impulso de todas las capas en 0,9, y seguir adelante.

*¿Cómo deberíamos probar nuestro nuevo impulso? Podríamos usar el MNIST conjunto de datos que utilizamos anteriormente, pero no es un buen candidato porque es demasiado fácil. Incluso una red simple y completamente conectada logra una precisión superior al 97 por ciento. Por lo tanto, reemplazaremos el conjunto de datos de dígitos MNIST por otro similar conjunto de datos que se sabe que representa un desafío mayor: el conjunto de datos Fashion-MNIST. (Consulte "Fashion-MNIST: un nuevo conjunto de datos de imágenes para máquinas de evaluación comparativa). Algoritmos de aprendizaje" de Han Xiao et al., arXiv:1708.07747 [2017].)*

El conjunto de datos Fashion-MNIST (FMNIST) es un sustituto directo del conjunto de datos MNIST existente. Contiene imágenes de 10 clases de ropa, todas Escala de grises de 28×28 píxeles. Para nuestros propósitos, haremos lo mismo que hicimos para MNIST y reduzca las imágenes de 28 × 28 píxeles a 14 × 14 píxeles. Las imágenes están en el conjunto de datos. directorio como matrices NumPy. Entrenemos un modelo usándolos. El código para el El modelo es similar al del Listado 10-7, excepto que en el Listado 11-7 reemplazamos el Conjunto de datos MNIST con FMNIST:

---

```
x_train = np.load("fmnist_train_images_small.npy")/255
x_test = np.load("fmnist_test_images_small.npy")/255
y_train = np.load("fmnist_train_labels_vector.npy")
y_test = np.load("fmnist_test_labels.npy")
```

---

Listado 11-7: Cargando el conjunto de datos Fashion-MNIST

También incluimos código para calcular el coeficiente de correlación de Matthews (MCC) en los datos de la prueba. Nos encontramos por primera vez con el MCC en el Capítulo 4, donde Aprendí que es una mejor medida del desempeño de un modelo que la precisión. El código a ejecutar está en fmnist.py. Tomando alrededor de 18 minutos en un caja Intel i5 más antigua, se produjo una ejecución

---

[[866 1 14 28 8 [ 5 958 2	1 68 0 14	0]
25 5 0 3 [ 20 1 790 14 126 0 44 1	0	2 0]
[ 29 21 15 863 46		3 1]
	1 20 0	5 0]

---

```
[0 0 91 22 849 1 32 0 0]
[0 0 0 1 0 960 0 22 2 15]
[161 2 111 38 115 0 556 0 17 0]
[0 0 0 0 0 29 0 942 0 29]
[1 0 7 6 4 973 0] 5      2      2
[0 0 0 0 0 6 0 29 1 964]]
```

5

precisión = 0,8721000  
MCC = 0,8584048

---

La matriz de confusión, todavía de  $10 \times 10$  debido a las 10 clases en FM-NIST, es bastante ruidosa en comparación con la matriz de confusión muy limpia que vimos. con MNIST propiamente dicho. Este es un conjunto de datos desafiante para modelos totalmente conectados. Recordemos que el MCC es una medida donde cuanto más cerca esté de uno, mejor el modelo.

La matriz de confusión anterior es para un modelo entrenado sin impulso. La tasa de aprendizaje fue de 1,0 y se entrenó para 40.000 minibatches. ¿Qué sucede si agregamos un impulso de 0,9 a cada capa completamente conectada y reducimos la tasa de aprendizaje a 0,2? Cuando añadimos impulso, Tiene sentido reducir la tasa de aprendizaje para no dar grandes pasos. agravado por el impulso que ya se está moviendo en una dirección particular. Explore qué sucede si ejecuta fmnist.py con una tasa de aprendizaje de 0,2 y sin impulso.

La versión del código con impulso se encuentra en fmnist\_momentum.py. Después de unos 20 minutos, se produjo una ejecución de este código.

```
[[766 5 14 61 2 0]      1 143 0      8
 [1 958 2 30 3 0 6 0]          0      0
 [ 12 0 794 16 98 0 80 0 0 ]      0
 [8 11 13 917 21 0 27 0 0]      3
 [0 0 84 44 798 0 71 0 0]      3
 [0 0 0 1 0 938 0 31 1 29]
 [76 2 87 56 60 0 714 0 0]      5
 [0 0 0 0 0 11 0 963 0 26]
 [ 1 1 6 8 1 10 4 964 0]5
 [0 0 0 0 0 6 0 33 0 961]]
```

precisión = 0,8773000  
MCC = 0,8638721

---

dándonos un MCC ligeramente más alto. ¿Eso significa que el impulso ayudó? Tal vez. Como ya sabemos, el entrenamiento de redes neuronales es un proceso estocástico. Por lo tanto, no podemos confiar en los resultados de un único entrenamiento de los modelos. Nosotros Es necesario entrenar los modelos muchas veces y realizar pruebas estadísticas sobre los resultados. ¡Excelente! Esto nos da la oportunidad de hacer un buen uso de los conocimientos sobre pruebas de hipótesis que adquirimos en el capítulo 4.

En lugar de ejecutar fmnist.py y fmnist\_momentum.py una vez cada uno, ejecútelo 22 veces cada uno. Esto lleva la mayor parte del día en mi antiguo Intel

Sistema i5, pero la paciencia es una virtud. El resultado neto es 22 valores de MCC para el modelo con impulso y 22 para el modelo sin impulso. No hay nada mágico en 22 muestras, pero pretendemos utilizar la prueba U de Mann-Whitney, y la regla general para esa prueba es tener al menos 20 muestras en cada conjunto de datos.

La Figura 11-8 muestra histogramas de los resultados.

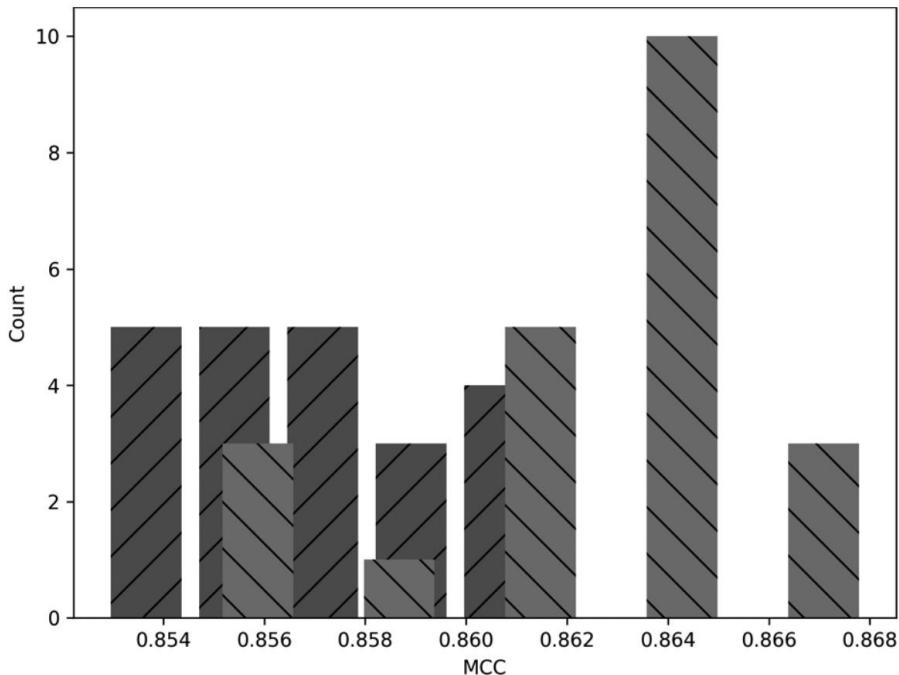


Figura 11-8: Histogramas que muestran la distribución de MCC para modelos entrenados con impulso (gris claro) y sin impulso (gris oscuro)

Las barras grises más oscuras son los valores de MCC sin momento, y las más claras las barras son las que tienen impulso. Visualmente, los dos son muy distintos entre sí. El código que produce la Figura 11-8 se encuentra en el archivo `fmnist_analyze.py`. Eche un vistazo al código. Utiliza `ttest_ind` y `mannwhitneyu` de SciPy junto con la implementación que proporcionamos en el Capítulo 4 de la `d` de Cohen para calcular el tamaño del efecto. Los valores de MCC en sí están en los archivos NumPy enumerados en el código.

Junto con el gráfico, `fmnist_analyze.py` produce el siguiente resultado:

---

```
sin impulso: 0,85778 +/- 0,00056 impulso:  
0,86413 +/- 0,00075
```

---

```
Impulso de la prueba t vs no (t,p): (6.77398299, 0.00000003):  
Mann-Whitney U          (41.00000000, 0.00000126)  
Cohen d                 : 2.04243
```

---

donde las dos líneas superiores son la media y el error estándar de la media.

Los resultados de la prueba t son (t, p), el estadístico de la prueba t y el valor p asociado. Similarmente,

los resultados de la prueba U de Mann-Whitney son ( $U, p$ ), el estadístico U y su valor  $p$ . Recuerde que la prueba U de Mann-Whitney es una prueba no paramétrica que no supone nada sobre la forma de la distribución de los valores de MCC. La prueba t supone que están distribuidos normalmente. Como sólo tenemos 22 muestras cada una, realmente no podemos hacer ninguna declaración definitiva sobre si los resultados se distribuyen normalmente; los histogramas no se parecen mucho a las curvas gaussianas. Es por eso que incluimos los resultados de la prueba Mann-Whitney U.

Un vistazo a los valores  $p$  respectivos nos dice que la diferencia de medias entre los valores de MCC con y sin impulso es altamente estadísticamente significativa a favor de los resultados con impulso. El valor  $t$  es positivo y el resultado con impulso fue el primer argumento. ¿Qué pasa con el valor  $d$  de Cohen? Está un poco por encima de 2,0, lo que indica un tamaño del efecto (muy) grande.

¿Podemos decir ahora que el impulso ayuda en este caso? Probablemente. Produjo modelos de mejor rendimiento dados los hiperparámetros que utilizamos. La naturaleza estocástica del entrenamiento de redes neuronales hace posible que podamos modificar los hiperparámetros de ambos modelos para eliminar la diferencia que vemos en los datos que tenemos. La arquitectura entre los dos es fija, pero nada dice que la tasa de aprendizaje y el tamaño del minibatch estén optimizados para cualquiera de los modelos.

Un investigador puntilloso se sentiría obligado a ejecutar un proceso de optimización sobre los hiperparámetros y, una vez satisfecho de haber encontrado el mejor modelo para ambos enfoques, hacer una declaración más definitiva después de repetir el experimento. Afortunadamente, no somos investigadores puntillosos.

En su lugar, usaremos la evidencia que tenemos, junto con varias décadas de sabiduría adquirida por los investigadores de aprendizaje automático del mundo con respecto a la utilidad del impulso en el descenso de gradientes, para afirmar que, sí, el impulso ayuda a los modelos a aprender, y usted debe usarlo. en la mayoría de los casos.

Sin embargo, la cuestión de la normalidad exige una mayor investigación. Después de todo, buscamos mejorar nuestra intuición matemática y práctica con respecto al aprendizaje profundo. Por lo tanto, entrenemos el modelo con impulso para FMNIST, no 22 veces sino 100 veces. Como concesión, reduciremos el número de minibatches de 40.000 a 10.000. Aún así, espere pasar la mayor parte del día esperando a que finalice el programa. El código, que no analizaremos aquí, se encuentra en `fmnist_repeat.py`.

La Figura 11-9 presenta un histograma de los resultados.

Claramente, esta distribución no se parece en nada a una curva normal. La salida de `fmnist_repeat.py` incluye el resultado de la función `normaltest` de SciPy . Esta función realiza una prueba estadística sobre un conjunto de datos bajo la hipótesis nula de que los datos se distribuyen normalmente. Por lo tanto, un valor  $p$  inferior a, digamos, 0,05 o 0,01, indica datos que no se distribuyen normalmente. Nuestro valor  $p$  es prácticamente cero.

¿Qué hacer con la figura 11-9? En primer lugar, como los resultados ciertamente no son normales, no está justificado utilizar una prueba t. Sin embargo, también utilizamos la prueba no paramétrica U de Mann-Whitney y encontramos resultados estadísticamente muy significativos, por lo que nuestras afirmaciones anteriores siguen siendo válidas. En segundo lugar, la cola larga de la distribución de la figura 11-9 está hacia la izquierda. Incluso podríamos argumentar que el resultado es posiblemente bimodal: que hay dos picos, uno cerca de 0,83 y el otro, más pequeño, cerca de un MCC de 0,75.

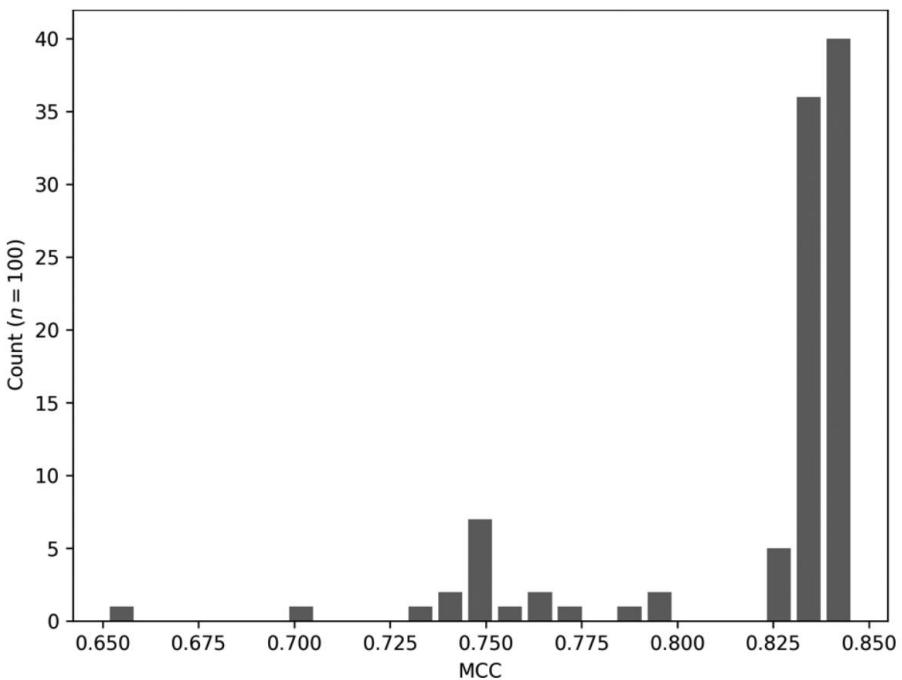


Figura 11-9: Distribución de valores de MCC para 100 entrenamientos del modelo FMNIST

La mayoría de los modelos se entrenaron a un nivel de rendimiento relativamente consistente, con un MCC cercano a 0.83. Sin embargo, la larga cola indica que cuando el modelo no era razonablemente bueno, era simplemente horrible.

Intuitivamente, la figura 11-9 me parece razonable. Sabemos que el descenso de gradiente estocástico es susceptible a una inicialización inadecuada, y nuestro pequeño conjunto de herramientas utiliza la inicialización de valores aleatorios pequeños de la vieja escuela. Parece probable que tengamos una mayor probabilidad de comenzar en una ubicación deficiente en el panorama de pérdidas y, después de eso, estemos condenados a un desempeño deficiente.

¿Y si la cola estuviera a la derecha? ¿Qué podría indicar eso? Una cola larga a la derecha significaría que el rendimiento de la mayoría de los modelos es de mediocre a pobre, pero, en ocasiones, aparece un modelo especialmente “brillante”. Tal escenario significaría que existen mejores modelos, pero que nuestra estrategia de entrenamiento y/o inicialización no es particularmente buena para encontrarlos. Creo que es preferible la cola de la izquierda: la mayoría de los modelos encuentran mínimos locales razonablemente buenos, por lo que la mayoría de los entrenamientos, a menos que sean horribles, terminan prácticamente en el mismo lugar en términos de rendimiento.

Ahora, examinemos una variante común del impulso, una que no conocerás. dudas que surjan durante su estancia en el aprendizaje profundo.

### El impulso de Nésterov

Muchos kits de herramientas de aprendizaje profundo incluyen la opción de utilizar el impulso de Nesterov durante el descenso del gradiente. El impulso de Nesterov es una modificación del descenso de gradiente ampliamente utilizada en la comunidad de optimización. La versión típicamente implementada en aprendizaje profundo actualiza el impulso estándar de

$$v \leftarrow v - f(x)$$

$$x \leftarrow x + v$$

a

$$v \leftarrow v - f(x + v)$$

$$x \leftarrow x + v$$

(11.6)

donde usamos notación de gradiente en lugar de parciales de una función de pérdida para indican que la técnica es general y se aplica a cualquier función,  $f(x)$ .

La diferencia entre el impulso estándar y el impulso de Nesterov de aprendizaje profundo es útil, solo un término que se agrega al argumento del degradado. La idea es utilizar el impulso existente para calcular el gradiente, no en la posición actual,  $x$ , sino en la posición de descenso del gradiente. sería si continuara usando el impulso actual,  $x + v$ . Entonces use el valor del gradiente en esa posición para actualizar la posición actual, como antes.

La afirmación, bien demostrada para la optimización en general, es que esto El ajuste conduce a una convergencia más rápida, lo que significa que el descenso del gradiente encontrará el mínimo en menos pasos. Sin embargo, aunque los kits de herramientas lo implementan, Hay razones para creer que el ruido que genera el descenso del gradiente estocástico con Los minibatches introducen compensaciones en el ajuste hasta el punto en que es poco probable que el impulso de Nesterov sea más útil para entrenar el aprendizaje profundo. modelos que el impulso regular. (Para más información sobre esto, vea el comentario en página 292 de Deep Learning de Ian Goodfellow et al.)

Sin embargo, el ejemplo 2D de este capítulo utiliza la función real para calcular calcular gradientes, por lo que podríamos esperar que el impulso de Nesterov sea efectivo en ese caso. Actualicemos el ejemplo 2D, minimizando la suma de dos gaussianas invertidas y veamos si el impulso de Nesterov mejora la convergencia, como reclamado. El código que ejecutaremos está en `gd_nesterov.py` y es prácticamente idéntico a el código en `gd_momentum.py`. Además, modifiqué un poco ambos archivos para devuelve la posición final después de que se completa el descenso del gradiente. De esa manera, nosotros Podemos ver qué tan cerca estamos de los mínimos conocidos.

La implementación de la Ecuación 11.6 es sencilla y afecta sólo la actualización de la velocidad, lo que provoca

$$vx = mu * vx - eta * dx(x,y)$$

$$vy = mu * vy - eta * dy(x,y)$$

convertirse

$$vx = mu * vx - eta * dx(x + mu * vx, y)$$

$$vy = mu * vy - eta * dy(x, y + mu * vy)$$

para sumar el impulso de cada componente,  $x$  e  $y$ . Todo lo demás se mantiene igual.

La Figura 11-10 compara el impulso estándar (arriba, de la Figura 11-7) y el impulso de Nesterov (abajo).

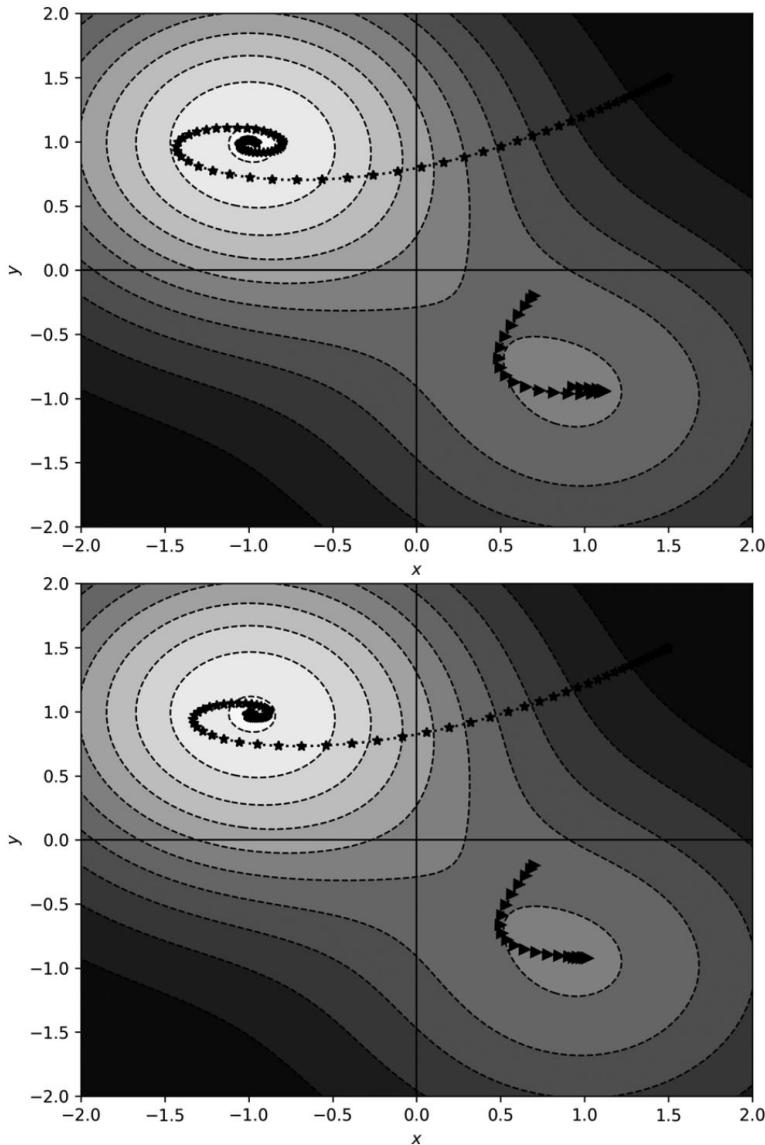


Figura 11-10: Momento estándar (arriba) e impulso de Nesterov (abajo)

Visualmente, el impulso de Nesterov muestra menos sobrepuerto, especialmente para la espiral que marca el camino que comienza en  $(1,5, 1,5)$ . ¿Qué pasa con la ubicación final que devuelve cada enfoque? Obtenemos la Tabla 11-2.

Tabla 11-2: Ubicación final para el descenso de gradiente con y sin  
El impulso de Nésterov

Punto inicial Estándar	Nésterov	Mínimo
(1,5,1,5)	(-0,9496, 0,9809) (-0,9718, 0,9813) (-1,1) (0,8807,	
(0,7,-0,2)	-0,9063) (0,9128, -0,9181) (1,-1)	

Los resultados del impulso de Nesterov están más cerca de los mínimos conocidos que los resultados del impulso estándar después del mismo número de pasos de descenso de gradiente.

## Descenso de gradiente adaptativo

El algoritmo de descenso de gradiente es casi trivial, lo que invita a la adaptación.

En esta sección, analizaremos las matemáticas detrás de tres variantes de descenso de gradientes populares entre la comunidad de aprendizaje profundo: RMSprop, Ada-grad y Adam. De los tres, Adán es el más popular con diferencia, pero vale la pena comprender los demás, ya que se construyen en sucesión hasta llegar a Adán. Estos tres algoritmos adaptan la tasa de aprendizaje sobre la marcha en alguna manera.

### RMSprop

Geoffrey Hinton presentó RMSprop, que significa propagación cuadrática media, en su serie de conferencias Coursera de 2012. Al igual que el impulso (con el que se puede combinar), RMSprop es un descenso de gradiente que rastrea el valor del gradiente a medida que cambia y utiliza ese valor para modificar el paso dado.

RMSprop utiliza un término de decaimiento ( $\gamma$ ) para calcular un promedio móvil de los gradientes a medida que avanza el algoritmo. En su conferencia, Hinton utiliza  $\gamma = 0,9$ .

La actualización del descenso de gradiente se vuelve

$$\text{metro} \leftarrow \text{metro} + (1 - \gamma)[f(x)]^2$$

$$v \leftarrow -\frac{\text{metro}}{\sqrt{\text{metro}}}$$

$$x \leftarrow x + v$$

(11.7)

Primero, actualizamos  $m$ , el promedio móvil de los cuadrados de los gradientes, ponderado por  $\beta$ , el término de caída. Luego viene el término de velocidad, que es casi el mismo que en el descenso de gradiente básico, pero dividimos la tasa de aprendizaje por la raíz cuadrada del promedio móvil, de ahí la parte RMS de RMSprop. Luego restamos la velocidad escalada de la posición actual para dar el paso.

Estamos escribiendo el paso como una suma, similar a las ecuaciones de momento anteriores (Ecuaciones 11.5 y 11.6); observe el signo menos antes de la actualización de velocidad.

RMSprop también funciona con impulso. Por ejemplo, extendiendo RMSprop con impulso de Nesterov es sencillo:

$$m \leftarrow m + (1 - \beta) [f(x + v)]^2$$

$$v \leftarrow v - \frac{f(x + v)}{\sqrt{m}}$$

$$x \leftarrow x + v \quad (11.8)$$

con el factor de impulso, como antes.

Se afirma que RMSprop es un clasificador robusto. Veremos a continuación cómo lo fue en una prueba. La consideramos una técnica adaptativa porque la tasa de aprendizaje ( $\eta$ ) se escala según la raíz cuadrada de la media del gradiente en ejecución; por lo tanto, la tasa de aprendizaje efectiva se ajusta en función del historial del descenso; no se fija de una vez por todas.

RMSprop se utiliza a menudo en el aprendizaje por refuerzo, la rama del aprendizaje automático que intenta aprender a actuar. Por ejemplo, jugar videojuegos de Atari utiliza el aprendizaje por refuerzo. Se cree que RMSprop es sólido cuando el proceso de optimización no es estacionario, lo que significa que las estadísticas cambian con el tiempo. Por el contrario, un proceso estacionario es aquel en el que las estadísticas no cambian con el tiempo. El entrenamiento de clasificadores que utilizan el aprendizaje supervisado es estacionario, ya que el conjunto de entrenamiento, por lo general, es fijo y no cambia, al igual que los datos que se envían al clasificador a lo largo del tiempo, aunque eso es más difícil de aplicar. En el aprendizaje por refuerzo, el tiempo es un factor y las estadísticas del conjunto de datos pueden cambiar con el tiempo; por lo tanto, el aprendizaje por refuerzo podría implicar una optimización no estacionaria.

## Adagrad y Adadelta

Adagrad apareció en 2011 (consulte “Métodos adaptativos de subgradiente para Learning and Stochastic Optimization” de John Duchi et al., Journal of Machine Learning Research 12[7], [2011]). A primera vista, parece bastante similar a RMSprop, aunque existen diferencias importantes.

Podemos escribir la regla de actualización básica para Adagrad como

$$v_i \leftarrow \frac{-f(x)_i}{\sqrt{\sum [f(x)_i^2]}} \\ x \leftarrow x + v$$

(11.9)

Esto requiere alguna explicación.

Primero, observe el subíndice  $i$  en la actualización de velocidad, tanto en la velocidad,  $v$ , como en el gradiente,  $f(x)$ . Aquí,  $i$  se refiere a un componente de la velocidad, lo que significa que la actualización debe aplicarse por componente. La parte superior de la ecuación 11.9 se repite para todos los componentes del sistema. Para una neurona profunda red, esto significa todos los pesos y sesgos.

A continuación, observe la suma en el denominador de la actualización de velocidad por componente. Aquí, ( $\tau$ ) es un contador de todos los pasos de gradiente tomados durante El proceso de optimización, es decir, para cada componente del sistema, Ada-grad rastrea la suma del cuadrado del gradiente calculado en cada paso. Si Estamos usando la Ecuación 11.9 para el undécimo paso de descenso de gradiente, luego la suma en el denominador tendrá 11 términos, y así sucesivamente. Como antes, es un aprendizaje. tasa, que aquí es global para todos los componentes.

También se utiliza ampliamente una variante de Adagrad: Adadelta. (Ver “Adadelta: Un método de tasa de aprendizaje adaptativo” por Matthew Zeiler, [2012].) Adadelta reemplaza la raíz cuadrada de la suma de todos los pasos en la actualización de velocidad con un promedio móvil de los últimos pasos, muy parecido al promedio móvil de RMSprop. Adadelta también reemplaza la tasa de aprendizaje global seleccionada manualmente, con un promedio móvil de las pocas actualizaciones de velocidad anteriores. Esto elimina la selección de un parámetro apropiado pero introduce un nuevo parámetro, para establecer el tamaño de la ventana, como se hizo para RMSprop. Es probable que sea menos sensible a las propiedades del conjunto de datos que es. Observe cómo en el original Papel Adadelta, se escribe como ( $\rho$ ).

## Adán

Kingma y Ba publicaron Adam, a partir de la “estimación del momento adaptativo”, en 2015, y ha sido citado más de 66.000 veces al momento de escribir este artículo. Adán usa el cuadrado del gradiente, como lo hacen RMSprop y Adagrad, pero también pista un término similar a un impulso. Presentemos las ecuaciones de actualización y luego caminemos a través de ellos:

$$\text{metro} \leftarrow 1 \text{ metro} + (1 - \frac{1}{t}) f(x)$$

$$v \leftarrow 2v + (1 - \frac{2}{t}) [f(x)]^2$$

$$m \leftarrow \frac{\text{metro}}{1 - \frac{t}{2}}$$

$$v \leftarrow \frac{b}{1 - \frac{t}{2}}$$

$$x \leftarrow x - \frac{\text{metro}}{\sqrt{v} + \epsilon} \quad (11.10)$$

Las dos primeras líneas de la ecuación 11.10 definen  $mv$  como promedios móviles del primer y segundo momento. El primer momento es el medio; el segundo momento es similar a la varianza, que es el segundo momento de la diferencia entre un punto de datos y la media. Observe la elevación al cuadrado del valor del gradiente en la definición de  $v$ . Los momentos de carrera están ponderados por dos parámetros escalares,  $1 - \frac{1}{t}$ .

Las dos líneas siguientes definen  $m$  y  $v$ . Estos son términos de corrección de sesgo para hacer que  $m$  y  $v$  sean mejores estimaciones del primer y segundo momento. Aquí,  $t$ , un entero que comienza en cero, es el paso de tiempo.

El paso real actualiza  $x$  restando el primer momento corregido por el sesgo,  $m$ , escalado por la relación de la tasa de aprendizaje global, y el cuadrado raíz del segundo momento corregido por el sesgo,  $v$ . El término es una constante para Evite la división por cero.

La ecuación 11.10 tiene cuatro parámetros, lo que parece excesivo, pero tres de ellos son fáciles de establecer y rara vez se modifican. El artículo original = 0,999 y = 10-8. Por lo tanto, al igual que seleccionar.  $\beta_2$  con vainilla = 0,9, sugiere 1 descenso de gradiente, el usuario debe Por ejemplo, Keras por defecto = 0,001, lo que funciona bien en muchos casos.

El artículo de Kingma y Ba muestra mediante experimentos que Adam generalmente supera al SGD con el impulso de Nesterov, RMSprop, Adagrad y Ada-delta. Probablemente esta sea la razón por la que Adam es actualmente el optimizador de referencia para muchas tareas de aprendizaje profundo.

#### Algunas reflexiones sobre los optimizadores

Qué algoritmo de optimización utilizar y cuándo depende del conjunto de datos. Como se mencionó, actualmente Adam es el favorito para muchas tareas, aunque el SGD correctamente ajustado también puede ser bastante efectivo, y algunos lo confían. Si bien no es posible hacer una declaración general sobre cuál es el mejor algoritmo, ya que no existe tal cosa, podemos realizar un pequeño experimento y discutir los resultados.

Este experimento, del que solo presentaré los resultados, entrenó una pequeña red neuronal convolucional en MNIST utilizando 16,384 muestras aleatorias para el conjunto de entrenamiento, un minilote de 128 y 12 épocas. Los resultados muestran la media y el error estándar de la media para cinco ejecuciones de cada optimizador: SGD, RMSprop, Adagrad y Adam. Es de interés la precisión del conjunto de pruebas y el tiempo del reloj de entrenamiento. Entrené todos los modelos en la misma máquina, por lo que debemos considerar el tiempo relativo. No se utilizó GPU.

La Figura 11-11 muestra la precisión general del conjunto de pruebas (arriba) y el tiempo de entrenamiento (abajo) por optimizador.

En promedio, SGD y RMSprop fueron aproximadamente un 0,5 por ciento menos precisos que los otros optimizadores, y RMSprop varió ampliamente pero nunca coincidió con Adagrad o Adam. Podría decirse que Adam tuvo el mejor desempeño en términos de precisión. En cuanto al tiempo de entrenamiento, SGD fue el más rápido y Adam el más lento, como podríamos esperar, dados los múltiples cálculos por paso que Adam realiza en relación con la simplicidad de SGD. En general, los resultados respaldan la intuición de la comunidad de que Adam es un buen optimizador.

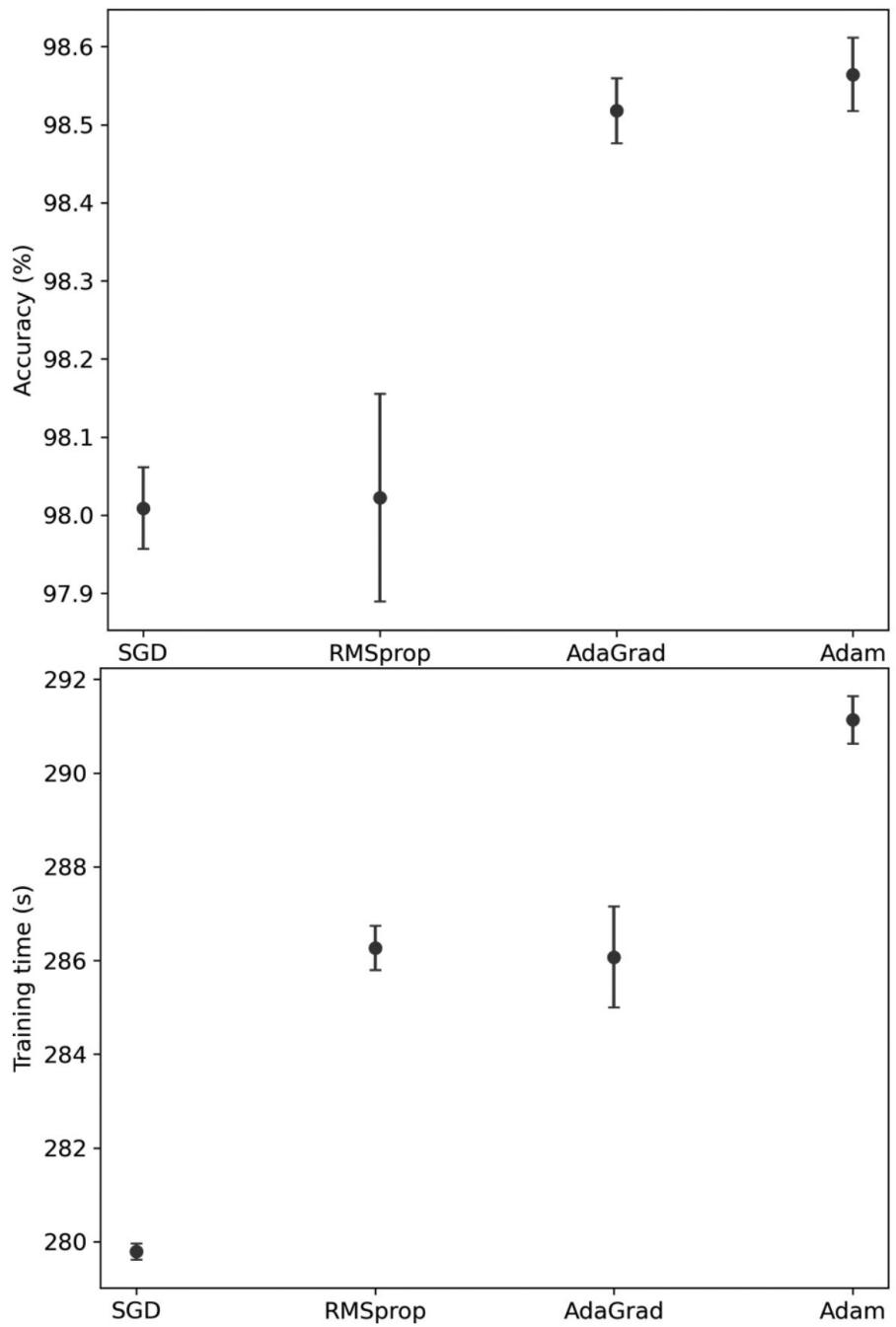


Figura 11-11: Precisión del modelo MNIST (arriba) y tiempo de entrenamiento (abajo) por optimizador

## Resumen

Este capítulo presentó el descenso de gradiente, trabajando a través de la forma básica, descenso de gradiente básico, con ejemplos 1D y 2D. Seguimos introduciendo el descenso de gradiente estocástico y justificamos su uso en el aprendizaje profundo.

A continuación analizamos el impulso, tanto de Standard como de Nesterov. Con el impulso estándar, demostramos que ayuda a entrenar modelos profundos (bueno, relativamente "profundos"). Mostramos visualmente el efecto del impulso de Nesterov utilizando un ejemplo 2D y discutimos por qué el impulso de Nesterov y el descenso del gradiente estocástico podrían contrarrestarse entre sí.

El capítulo concluyó con una mirada a la ecuación de actualización del descenso de gradiente para algoritmos avanzados, ilustrando así cómo el descenso del gradiente de vainilla invita a la modificación. Un experimento simple nos dio una idea de cómo funcionan los algoritmos y pareció justificar la creencia de la comunidad de aprendizaje profundo en la idoneidad general de Adam sobre SGD.

Y, con este capítulo, nuestra exploración de las matemáticas del aprendizaje profundo. ing llega a su fin. Todo lo que queda es un apéndice final que le indica los lugares donde puede acudir para obtener más información.

## Epílogo

Como dijo el gran científico informático Edsger W. Dijkstra: "No deberían existir las matemáticas aburridas". Sinceramente espero que no hayas encontrado este libro aburrido. Odiaría ofender al fantasma de Dijkstra. Si todavía estás leyendo en este punto, sospecho que encontraste algo de mérito. ¡Bien! Gracias por seguir adelante. Las matemáticas nunca deberían ser aburridas.

Hemos cubierto los conceptos básicos de lo que necesita para comprender y trabajar con el aprendizaje profundo. Sin embargo, no se detenga aquí: utilice las referencias del Apéndice y continúe con sus exploraciones matemáticas. Nunca debe estar satisfecho con su base de conocimientos; trate siempre de ampliarla.

Si tiene preguntas o comentarios, comuníquese conmigo a [mathfordeeplearning@gmail.com](mailto:mathfordeeplearning@gmail.com).



## IR MÁS LEJOS



El objetivo de este libro era analizar las matemáticas fundamentales detrás del aprendizaje profundo, el tipo de matemáticas necesarias para seguir lo que se hace en profundidad.

Qué es el aprendizaje y cómo funciona. Hemos hecho precisamente eso en los 11 capítulos anteriores.

En este apéndice, mi objetivo es indicarle más cosas. Por necesidad, Sólo chapoteamos en las pozas de marea, que son bastante fascinantes, pero en las profundidades encontrarás aún más belleza y elegancia. Lo que sigue son sugerencias que le ayudarán a sacar más provecho de los temas que cubrimos.

### Probabilidades y estadísticas

Hay cientos, si no miles, de libros sobre probabilidad y estadística. La lista aquí es, naturalmente, incompleta y no exhaustiva, pero debería ayudarle a ampliar sus conocimientos en estas áreas.

Probabilidad y estadística por Michael Evans y Jeffrey Rosenthal Un enfoque integral de libro de texto que está disponible de forma gratuita aquí: <http://www.utstat.toronto.edu/mikeevans/jeffrosenthal/book.pdf>. El libro de Evans y Rosenthal se dirige a lectores con exactamente el tipo de antecedentes que cubre este libro.

Estadística bayesiana de forma divertida por Will Kurt Analizamos el teorema de Bayes en el capítulo 3. Este libro presenta la estadística bayesiana de una manera accesible. La estadística bayesiana está fuertemente relacionada con el aprendizaje automático y Eventualmente lo encontrarás a medida que avances en tus estudios.

Introducción a la probabilidad por Joseph Blitzstein y Jessica Hwang

Otra introducción popular a la probabilidad, que incluye Monte

Carlos modelando.

Python para probabilidad, estadística y aprendizaje automático por José Unpingco Este libro proporciona una visión alternativa al enfoque que yo tomó en este libro. Cubre temas ligeramente diferentes, pero sigue usando Python y NumPy. La parte de aprendizaje automático cubre lo que yo llamo “aprendizaje automático clásico” con alguna mención al aprendizaje profundo.

Estadística práctica para la investigación médica de Douglas Altman Un clásico texto, pero sigue siendo muy legible y relevante, aunque proviene de la época antes se hacía mucho en computadoras personales. El enfoque es la bioestadística, pero lo básico es lo básico independientemente del área de aplicación.

## Álgebra lineal

De todos los temas que cubrimos, fuimos los más injustos con el álgebra lineal. El Las referencias aquí le ayudarán a apreciar toda la elegancia del tema.

Introducción al álgebra lineal por Gilbert Strang Un libro introductorio popular. Cubre con mayor detalle muchos de los temas que toqué en Capítulos 5 y 6.

Álgebra lineal de David Cherney et al. Similar al anterior, pero disponible de forma gratuita: <https://www.math.ucdavis.edu/~linear/linear-guest.pdf>.

Álgebra lineal de Jim Hefferon También disponible de forma gratuita y en el mismo nivel que los demás: <http://joshua.smcvt.edu/linearalgebra/book.pdf>.

## Cálculo

Hablamos del cálculo en los capítulos 7 y 8, pero nos limitamos a la diferenciación únicamente. El cálculo es, por supuesto, mucho más que diferenciación. El Otra parte principal del cálculo es la integración. Ignoramos la integración porque el aprendizaje profundo rara vez la utiliza. La convolución es integración cuando se trabaja con variables continuas, pero la mayoría de las integrales se convierten en sumatorias en el mundo digital. Las referencias enumeradas aquí llenarán los vacíos en nuestro superficial tratamiento.

Libro de trabajo de práctica de habilidades esenciales de cálculo de Chris McMullen This Un libro de texto/libro de trabajo popular cubre la diferenciación y el comienzo de la integración. Incluye soluciones a problemas. Vea este libro como una reseña de Capítulo 7 y una introducción a la integración.

Cálculo de James Stewart Si el libro de McMullen es una introducción amable, este libro es un tratamiento integral del tema. El libro cubre la diferenciación y la integración, incluido el cálculo multivariado, es decir, las derivadas parciales y el cálculo vectorial (consulte el Capítulo 8), y ecuaciones diferenciales. Incluye aplicaciones.

Cálculo diferencial matricial con aplicaciones en estadística y econometría por Jan Magnus y Heinz Neudecker Considerado por algunos como la referencia estándar de cálculo matricial, que proporciona un tratamiento profundo y completo del tema.

The Matrix Cookbook de Kaare Brandt Petersen y Michael Syskind Pedersen Una referencia popular para el cálculo matricial que va más allá de lo que cubrimos en el Capítulo 8. Puede encontrarlo aquí : <http://www2.imm.dtu.dk/pubdb/doc/imm3274.pdf>.

## Aprendizaje profundo

El aprendizaje profundo está evolucionando rápidamente. Si bien algunos de los primeros e impresionantes resultados del tipo “Guau, ni siquiera sabía que eso era posible” se están volviendo menos comunes, el campo está madurando silenciosamente y arraigándose en casi todas las áreas de la ciencia y la tecnología. Nuestro mundo nunca volverá a ser el mismo gracias a lo que el aprendizaje profundo ha hecho posible.

Aprendizaje profundo de Ian Goodfellow, Yoshua Bengio y Aaron Courville Uno de los primeros libros de texto específicos de aprendizaje profundo y ampliamente considerado como uno de los mejores. Cubre todos los elementos esenciales, pero a veces va bastante rápido.

Un enfoque de álgebra matricial para la inteligencia artificial por Xian-Da Zhang Un nuevo libro que cubre tanto las matrices como el aprendizaje automático, incluido el aprendizaje profundo. Véalo como un tratamiento más matemático del aprendizaje automático.

Especialización en aprendizaje profundo en Coursera No es un texto, sino una serie de cursos en línea con instructores de primer nivel. Puede encontrar los cursos aquí: <https://www.coursera.org/specializations/deep-learning/>.

Conferencias de Coursera de Geoffrey Hinton En 2012, Hinton dio una serie de conferencias sobre Coursera, y vale la pena escucharlas incluso ahora. RMSprop se analizó en esta serie. Las conferencias son bastante accesibles y no demasiadas matemáticas. Puede encontrarlos aquí: [https://www.cs.toronto.edu/~hinton/coursera\\_lectures.html](https://www.cs.toronto.edu/~hinton/coursera_lectures.html) .

Aprendizaje profundo: un enfoque visual por Andrew Glassner Este libro cubre la amplitud del aprendizaje profundo, desde el aprendizaje supervisado hasta el aprendizaje por refuerzo, todo sin matemáticas. Úselo como una introducción rápida y de alto nivel a muchos temas en el campo.

Reddit Para mantenerse al tanto de la comunidad de aprendizaje profundo, siga la conversación en Reddit: <https://www.reddit.com/r/MachineLearning/>.

Arxiv Arxiv , que se encuentra en <https://arxiv.org/>, es un repositorio de preimpresión. El últimos artículos sobre aprendizaje profundo se mostrarán aquí. Tenga en cuenta que, dado que el campo avanza tan rápidamente, la publicación en revistas revisadas por pares no es la norma. Más bien, publicar artículos en arxiv.org, especialmente aquellos presentados en conferencias, es la forma de ver las últimas investigaciones. Arxiv se divide en categorías. Sin embargo, los que he proporcionado aquí son los que tiendo a seguir. hay otros:

- Visión por computadora y reconocimiento de patrones: <https://arxiv.org/list/cs.CV/reciente/>
- Inteligencia artificial: <https://arxiv.org/list/cs.AI/recent/>
- Computación neuronal y evolutiva: <https://arxiv.org/list/cs.NE/reciente/>
- Aprendizaje automático: <https://arxiv.org/list/stat.ML/recent/>

## ÍNDICE

### A

Adadelta, 299  
 Adagrad, 299  
 Adán, 300  
 transformación afín, 128 media  
 aritmética, 70  
 AutoML, 283

**B**  
 retropropagación, 244  
 algoritmo, 256  
 código  
     manual,  
     249 derivados, 247  
 gráfico computacional, 267  
 error, 255  
 red completamente conectada, 255  
     implementación, 260  
     pérdida,  
     255 símbolo a número, 268  
     símbolo a símbolo, 268  
 entrenamiento por lotes, 282  
 Teorema de Bayes, 21, 31  
     evidencia, 59  
     probabilidad, 59  
 Clasificador ingenuo de Bayes,  
     62 supuesto de independencia, 63  
 probabilidad posterior, 59  
 probabilidad previa, 60  
 anterior desinformado, 62  
 actualización del anterior, 61  
 Bayes, Thomas, 59  
 distribución beta, 53  
 función binomial , 47  
 paradoja del cumpleaños,  
 26 matriz de bloques,  
 125 diagrama  
     de caja,  
     80 volantes, 82 bigotes, 82

### C

regla de  
 cadena  
     derivada de cálculo,  
     170, 184  
     constante, 167 definición,  
     165, 166 direccional,  
     188 exponencial,  
     175  
     primero, 167  
     logaritmo, 176 parcial  
     mixto, 183  
     notación, 166  
     parcial, 181 regla  
     de potencia, 168  
     regla del producto,  
     169 regla del  
     cociente, 169  
     reglas (tabla), 176 segundos, 167  
 funciones  
     trigonométricas, 172  
     diferencial, 163 diferenciación,  
     167 extremos (extremo),  
     177 extremos  
         globales, 178  
     gradiente, 186 campos  
     vectoriales, 186  
     punto de  
         inflexión, 178 integral,  
         163 límite, 166 extremos locales, 178 cálculo matricial. Ver  
         máximos, 177  
         mínimos, 177  
     notación, 187  
     punto de silla, 178  
     campo escalar,  
     186 recta secante,  
     165  
     pendiente, 164 puntos  
     estacionarios, 165 recta tangente, 165  
 Producto cartesiano, 118  
 teorema del límite central, 55  
 centroide, 149

- regla de la cadena (derivadas), 170 , 184  
 regla de la cadena (probabilidad),  
     , 37 97  
 Combinaciones d de Cohen  
     (fórmula), 28 gráfico  
     computacional, 267 probabilidad  
     condicional, 31 intervalo de  
         confianza (CI), 96  
         cálculo, 97  
         interpretación, 97  
     valor crítico, 97 matriz de  
     confusión, 254 tablas de  
     contingencia, 254  
     diagrama de contorno, 276 convolución, 229  
         1D, 230  
         2D, 233  
             red neuronal (CNN), 234  
             correlación cruzada, 232  
             filtro, 235  
             kernel, 231  
             no linealidad, 237  
             zancada,  
                 234 relleno con ceros,  
             231 correlación  
                 Pearson, 86 años  
                 Spearman, 90  
         matriz de covarianza, 147  
     regla de la mano  
         derecha del producto cruzado, 119
- D
- diagrama de caja de  
     datos, 80 canales, 223  
 CIFAR-10, 224  
 conjunto de datos como matriz, 222  
 Fashion-MNIST, 290  
 espacio de funciones,  
     105 funciones,  
     105 intervalos,  
     68 datos faltantes, 83  
 MNIST, 239  
 nominal, 68 no  
 es un número (NaN), 83  
 codificación one-hot, 69  
 ordinal, 68  
 valores  
     atípicos, 82 ratio, 68
- forma, 225  
 resumen estadístico, 70  
 grados de libertad, 95 regla  
 de la cadena  
     derivada, 170 , 184  
     constante, 167  
     diferenciación, 167  
     direccional, 188  
     exponencial, 175  
     primero,  
         167 campo  
             vectorial gradiente,  
         186 logaritmo, 176  
     parcial mixto, 183  
     notación, 166  
     parcial, 181  
     regla de potencia,  
         168 regla del producto,  
         169 regla del cociente,  
         169 reglas (tabla),  
         176 campo escalar,  
         186 segundo,  
             167 funciones trigonométricas, 172  
     propiedades  
         determinantes,  
         134  
         desviación  
         media, 74  
     muestra, 75 sistema  
         autónomo de ecuaciones  
         diferenciales, 206  
     puntos críticos, 207 distribución (probabilidad), 41  
         Bernoulli, 48  
         beta, 53  
         binomial, 46  
         continua, 51  
         Rodillo de datos de carga rápida (FLDR),  
             49 gamma, 53  
             Gaussiano, 53  
             lognormal, 83  
             normal, 53 74 , 83  
             Poisson, 48  
         función de densidad de probabilidad,  
             53 uniforme, 45 51  
     producto escalar, 114
- mi
- tamaño del efecto,  
 97 valor propio, 141

- vector propio, 141
- Distancia euclídea, 146
- eventos, 18
- pruebas (bayesianas), 59
- F**
- Puntuación F1,  
72 falsos negativos (FN), 251  
falsos positivos (FP), 251  
Moda-MNIST, 290  
Rodillo de datos de carga rápida (FLDR), 49  
espacios de funciones,  
105 funciones,  
105 números de punto flotante, 5  
capas completamente conectadas, 239
- G**
- distribución gamma, 53  
Distribución gaussiana, 53 media  
geométrica, 71 campo  
vectorial  
degradado, 186  
descenso de gradiente  
Adadelta, 299  
Adagrad, 297, 299  
Adam, 297, 300  
entrenamiento por lotes,  
282 efecto de mínimos múltiples, 281 en  
1D, 272 en  
2D, 276  
minibatch, 283, 284  
impulso, 285 red  
neuronal, 289  
Impulso de Nesterov, 294  
aprendizaje en línea, 283  
RMSprop, 297  
estocástico, 282  
vainilla, 272
- h**
- Producto de Hadamard, 110  
media armónica, 72  
Matriz de Hesse, 211  
como jacobiana de gradiente, 212  
Descomposición de Cholesky, 217  
puntos críticos, 213  
optimización, 214  
aproximación cuadrática, 215
- Hinton, Geoffrey, 297  
conversión  
de histograma a probabilidades, 43  
definición, 43  
prueba de hipótesis, 92 valor  
p, 95 alfa, 96  
hipótesis  
alternativa, 94 suposiciones, 95  
calcular IC, 97  
intervalo de confianza  
(IC), 96 interpretación, 97 valor  
crítico, 97 grados de  
libertad, 95 hipótesis,  
94 interpretación, 94
- U**
- U de Mann-Whitney, 93, 99 no  
paramétrico, 93 hipótesis  
nula, 94 unilateral, 94  
paramétrico, 93,  
95 estadísticamente  
significativo, 96 prueba t, 93
- I**
- suposiciones, 95 bilateral,  
94 advertencia,  
96  
Prueba t de Welch, 95  
Prueba de suma de rangos de Wilcoxon, 99
- j**
- matriz identidad, 132  
matriz indefinida, 140 inercia,  
285 producto  
interno, 114 rango  
intercuartil, 82 matriz inversa,  
138
- j**
- probabilidad conjunta, 37  
tabla, 33
- k**
- k-vecinos más cercanos, 105  
Producto Kronecker, 125  
Divergencia Kullback-Leiber, 151

## I

- Norma L1, 145
- Norma L2, 145
- ley de grandes números, 58
- Hagamos un trato, 46
- probabilidad (bayesiana), 59
- definición de
  - álgebra lineal, 103
- Producto Hadamard, 110
- Divergencia de Kullback-Leibler, 151 matriz, 105
  - transformación afín, 128 bloque, 125 ecuación
  - característica, 142 polinomio
  - característico, 142 cofactor, 136
  - expansión de
  - cofactor, 136 transpuesta
  - conjugada, 140 covarianza, 147
  - determinante, 134
  - propiedades
  - determinantes, 134 producto directo, 125 valor propio , 141
  - vector propio, 141
- Hermitiano, 140
- Adjunto hermitiano, 140
- identidad, 132
- indefinido, 140
- inverso, 138
- Producto Kronecker, 125 menor, 135
- Pseudoinverso de Moore-Penrose, 160
  - multiplicación, 120, 121 negativo
  - definido, 140 negativo
  - semidefinido, 140 no degenerado, 138 no singular, 138 unos, 131 orden, 106
  - ortogonal,
  - 139 positivo
  - definido, 140 positivo
  - semidefinido, 140 rotación, 128 singular, 138 cuadrado, 123
  - simétrico, 139
  - traza , 130
  - transponer, 130
- triangular, 133 cero, 131 análisis
- de componentes principales (PCA), 154
- entropía relativa, 151
- escalar, 104
- descomposición de valores singulares (SVD), 157
- tensor
  - aritmética, 109
  - operaciones de matriz, 109
  - orden, 106
- vector
  - distancia del vagón, 146
  - centroide, 149
  - Distancia de Chebyshev, 146
  - distancia de cuadra de la ciudad, 146
  - columnas, 104 producto
  - cruzado, 119 producto escalar, 114
  - Distancia euclídea, 146 norma
  - infinita, 145 producto
  - interno, 114 magnitud, 112
  - Distancia de Mahalanobis, 148
  - Distancia de Manhattan, 146
  - norma, 144
  - ortogonal, 115
  - producto exterior, 117
  - proyección, 116
  - regla de la mano derecha, 119 fila, 104
  - Distancia del taxi, 146
  - transposición, 113
  - unidad, 112
  - función logística, 229
- METRO
  - Distancia de Mahalanobis, 148
  - Distancia de Manhattan, 146
  - Universidad Mann-Whitney, 99
  - hipótesis nula, 100
  - probabilidad marginal, 33
  - Matplotlib, 12
  - matrices, 105
    - transformaciones afines, 128
    - bloques, 125
    - cálculos. Véase ecuación característica
    - del cálculo matricial, 142.

- polinomio característico, 142 cofactor, 136 expansión de cofactor, 136 transpuesta conjugada, 140 covarianza, 147 propiedades determinantes, 134 producto directo, 125 valor propio, 141 vector propio, 141 Hermitiano, 140 Adjunto hermitiano, 140 identidad, 132 indefinido, 140 inverso, 138 Producto Kronecker, 125 menor, 135 Pseudoinverso de Moore-Penrose, 160 multiplicación, 121 propiedades, 120 negativo definido, 140 negativo semidefinido, 140 no degenerado, 138 no singular, 138 unos, 131 orden, 106 ortogonal, 139 positivo definido, 140 positivo semidefinido, 140 rotación, 128 singular, 138 cuadrado , 123 simétrico, 139 traza, 130 transpuesta, 130 triangular, 133 cero, 131 cálculo matricial, 193 regla de la cadena función escalar por matriz, 203 función escalar por vector, 200 función vectorial por escalar, 202 función vectorial por vector, 203 comparación jacobiana, gradientes, pendientes, 205 diseño del denominador, 194 función de activación derivada, 219 operaciones por elementos, 217
- Matriz de Hesse, 205, 211 como jacobiano de gradiente, 212 identidades función escalar por matriz, 203 función escalar por vector, 199 función vectorial por escalar, 202 función vectorial por vector, 203 Matriz jacobiana, 205 función matricial por escalar, 198 disposición del numerador, 194 función escalar por matriz, 198 función escalar por vector, 196 tabla de derivadas, 194 vector tangente, 196, 198 función vectorial por vector, 197 función con valores vectoriales, 195 producto directo matricial, 125 multiplicación de matrices, 121 propiedades, 120 significar aritmética, 70 geométrica, 71 armónica, 72 desviación media, 74 mediana, 72 desviación absoluta mediana, 76 minibatch, 283, 284 datos faltantes, 83 impulso, Nesterov, 294 Dilema de Monty Hall, 19, 46 eventos mutuamente excluyentes, 24
- Clasificador ingenuo de Bayes, 62 supuesto de independencia, 63 clasificador de centroide más cercano, 149 matriz definida negativa, 140 matriz semidefinida negativa, 140 red neuronal truco de sesgo, 129 vector de sesgo, 225 convolución, 229 1D, 230 capa convolucional, 234 filtro, 235 forma de conjunto de datos, 225 incrustación, 119 características, 105

- red neuronal, espacio de
  - características continuas,
  - 105 avance, 225 capa
  - completamente conectada, 239
  - hiperparámetro, 283
  - inicialización, 42
  - función logística, 229
  - minibatch, 224, 264 impulso,
  - 289 capa de agrupación,
  - 237 unidad lineal
  - rectificada (ReLU), 226 función sigmoidea,
  - 229 entrenamiento, 259 matriz
  - de peso, 225
  - neuroevolución, 217
- El método de Newton, 208.
  - Matriz de arpillera, 216
  - jacobiano, 209
    - Aproximación de la serie de Taylor, 215
  - distribución normal, 53 no es un número (NaN), 83
  - NumPy, 4
    - indexación de matrices,
    - 8 matrices en disco, 10
    - transmisiones, 110 dos puntos, 9
    - tipos de datos, 6
    - definiciones de matrices,
    - 5 puntos
    - suspensivos, 9 multiplicaciones de matrices, 123 matrices especiales, 7
- oh
  - codificación one-hot, 69
  - aprendizaje en línea, 283
  - optimización de
    - primer orden, 214, 215
    - neuroevolución, 217 de segundo orden, 214, 215
      - intratable, 217 matriz
    - ortogonal, 139 producto externo,
    - 117 valores atípicos, 82
- valor p, 95
  - análisis de componentes principales (PCA), 154
- promedio de la capa de agrupación, 238 pérdida de información, 238
- máximo, 237 matriz definida positiva, 140 matriz semidefinida positiva, 140 probabilidad posterior, 59 precisión, 72 análisis de componentes principales (PCA), 154 probabilidad anterior, 60 actualización, 61 probabilidad
- Teorema de Bayes, 21, 31
  - evidencia, 59
  - probabilidad, 59
  - probabilidad posterior, 59
  - probabilidad previa, 60
  - anterior desinformado, 62
  - actualización del anterior, 61
  - paradoja del cumpleaños, 26
  - teorema del límite central, 55 regla de la cadena, 37
  - condicional, 31
  - definición, 18
  - distribución, 41
    - Bernoulli, 48 beta, 53, 83 binomial, 46 continuo, 51
    - discreto, 45
- Rodillo de datos de carga rápida (FLDR), 49
  - del histograma, 44 gamma, 53
  - Gaussiano, 53
  - lognormal, 83
  - normal, 53, 83
  - Poisson, 48
  - función de densidad de probabilidad, 53
  - uniforme, 45, 51
- enumerando el espacio muestral, 23 evento, 18 conjunto, 17,
  - 33, 37 tabla de
  - probabilidad conjunta, 33 ley de grandes números, 58 marginal, 17, 33 eventos mutuamente excluyentes, 24

de un evento, regla	estadísticamente significativo, 12, 96
de 22 productos, regla	correlación
de 25 productos (condicional), 31 variable	estadística, 86
aleatoria	Pearson, 86 años
continuo, 19 discreto,	Spearman, 90
19 muestral, 18	definición, 67
espacio	grados de libertad, 95 desviación
muestral, 18 regla de	muestra
suma (eventos dependientes), 25 regla de	sesgada, 75 media, 74
suma (eventos independientes), 24	mediana
Titanic, 35 en	absoluta, 76 error estándar,
total, 32	77 muestra insesgada, 75
dos dados, 23	
probabilidad (distribución) beta, 83	Puntuación F1,
función de	72 prueba de hipótesis, 92 alfa,
densidad de probabilidad, 53 eventos	96 hipótesis
condicionales de regla del producto	alternativa, 94 suposiciones, 95
(probabilidad), 31 eventos	calcular IC, 97
independientes, 25	
PyTorch, 267	d de Cohen, 97
	intervalo de confianza (IC), 96 valor
q	crítico, 97 tamaño del
cuantiles, 78	efecto, 97 hipótesis,
cuartiles, 78	94 interpretación, 94
R	U de Mann-Whitney, 93, 99 no
variable aleatoria	paramétrica, 93 hipótesis
continuo, 19 discreto,	nula, 94 unilateral, 94 valor
19 recuerdo, 72	p, 95 paramétrica,
recursividad,	93, 95
135 aprendizaje por	estadísticamente
refuerzo, 298 entropía relativa, 151	significativa, 96 prueba t, 93, 95
	suposiciones de
RMSprop, matriz de	prueba t, 95 bilateral , 94
rotación 297, 128	advertencia, 96
RMSprop, matriz de	Prueba t de Welch, 95
rotación 297, 128	Prueba de suma de rangos de Wilcoxon,
	99 rango intercuartil, 82
S	Universidad Mann-Whitney, 292
punto de silla, 178	significar
muestra (probabilidad), 18 espacio	aritmética, 70
muestral, 18 escalar,	geométrica, 71
104 scikit-	armónica, 72
learn, 14	mediana, 70, 72 no
SciPy, 11	estacionaria, 298
función sigmoidea, 229 matriz	codificación one-hot, 69
singular, 138 descomposición	precisión, 72
de valor singular, 157 matriz cuadrada, 123 error	cuantiles, 78
estándar (de la media), 77	

estadística, cuartiles	Ud.
continuos, 78	
recuerdo,	
72 desviación estándar, 70	
error estándar, 70	
estacionario, 298	
resumen, 70	
prueba t,	
292 tipos de	
intervalo de	
datos, 68	
nominal, 68	
ordinal, 68	
relación, 68	
varianza, 70 muestra	
sesgada, 75 mediana	
absoluta, 76 error	
estándar, 77 muestra	
insesgada, 75 descenso de gradiente	
estocástico, 282 regla de suma (probabilidad)	
eventos dependientes, 25	
eventos independientes, 24	
Ciudades sumerias, 32	
estadísticas resumidas, 70	
optimización de enjambre, 217	
matriz simétrica, 139	
 t	
prueba t	
supuestos, 95	
intervalo de confianza, 97	
Welch, 95	
Serie de Taylor, 214	
tensor, 106	
operaciones	
de matriz aritmética, 109	
orden, 106	
TensorFlow, 267 kits	
de	
herramientas, 2	
probabilidad	
total, 32 traza, 130	
entrenamiento de una red, 259 función	
trascendental, 214 matriz	
triangular, 133 verdadero	
negativo (TN), 254 verdadero	
positivo (TP), 254 tablas 2 × 2, 254	
	V
	muestra
	sesgada de varianza,
	75 mediana absoluta, 76
	error estándar, 77
	muestra insesgada, 75
	vector
	distancia del vagón, 146
	centroide, 149
	Distancia de Chebyshev, 146
	distancia de cuadra de la
	ciudad, 146
	columnas, 104 producto
	cruzado, 119 producto escalar, 114
	Distancia euclídea, 146 norma
	infinita, 145 producto
	interno, 114 notación
	matricial, 123
	Norma L1, 145
	Norma L2,
	magnitud 145, 112
	Distancia de Mahalanobis, 148
	Distancia de Manhattan, 146
	norma, 144
	ortogonal, 115
	producto exterior, 117
	notación matricial, 123
	proyección, 116
	regla de la mano derecha,
	119 fila, 104
	Distancia del taxi, 146
	transposición, 113
	unidad, 112
	 W.
	Prueba t de Welch, 95
	Prueba de suma de rangos de Wilcoxon, 99



Nunca antes el mundo había dependido tanto de Internet para mantenerse conectado e informado. Eso hace que la misión de Electronic Frontier Foundation (garantizar que la tecnología respalte la libertad, la justicia y la innovación para todas las personas) más urgente que nunca.

Durante más de 30 años, la EFF ha luchado por los usuarios de tecnología a través del activismo, en los tribunales y desarrollando software para superar los obstáculos a su privacidad, seguridad y libertad de expresión. Esta dedicación nos fortalece a todos a través de la oscuridad. Con su ayuda podemos navegar hacia un futuro digital más brillante.



[APRENDA MÁS Y ÚNASE A EFF EN EFF.ORG/NO-STARCH-PRESS](http://EFF.ORG/NO-STARCH-PRESS)

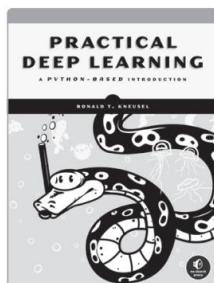
# RECURSOS

Visite <https://nostarch.com/math-deep-learning/> para obtener erratas y más información.

Más libros sensatos de



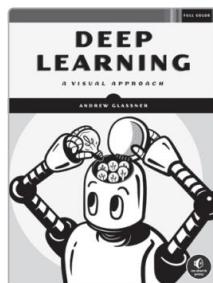
SIN PRENSA DE ALMIDÓN



APRENDIZAJE PROFUNDO PRÁCTICO

Una introducción basada en Python  
por ronald t. arrudarise

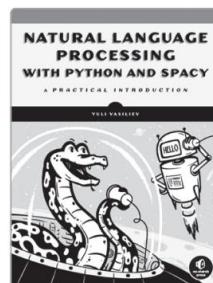
464 pp., \$59.95 isbn  
978-1-71850-074-7



APRENDIZAJE PROFUNDO

Un enfoque visual  
por andrew glassner

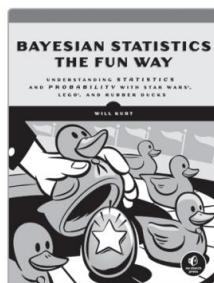
776 pp., \$99.99 isbn  
978-1-71850-072-3



PROCESAMIENTO NATURAL DEL LENGUAJE

CON PYTHON Y ESPACIOSO  
Una introducción práctica

por yuli vasilev  
216 pers., \$39.95  
isbn 978-1-71850-052-5

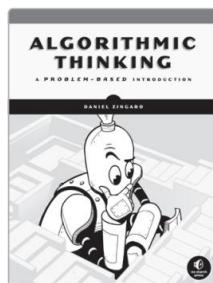


ESTADÍSTICAS BAYESIANAS

LA MANERA DIVERTIDA

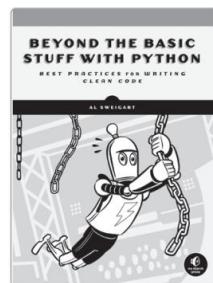
Comprensión de la estadística y la probabilidad con  
Star Wars, LEGO y patitos de goma  
por kurt

256 pers., \$34.95  
isbn 978-1-59327-956-1



PENSAMIENTO ALGORÍTMICO

Una introducción basada en problemas  
por daniel zingaro  
408 pers., \$49.95  
isbn 978-1-71850-080-8



MÁS ALLÁ DE LO BÁSICO

CON PITÓN

Mejores prácticas para escribir código limpio  
por al swigart  
384 pp., \$34.95 isbn  
978-1-59327-966-0

teléfono:  
800.420.7240 o  
415.863.9900

correo electrónico:  
[ventas@nostarch.com](mailto:ventas@nostarch.com)  
web:  
[www.nostarch.com](http://www.nostarch.com)

Fo reformulado  
y  
OORDDEREQJ WA LV  
DOCTOR



# EL BÁSICO MATEMÁTICAS PARA IA , EXPLICADO

Para comprender verdaderamente el poder del aprendizaje profundo, es necesario comprender los conceptos matemáticos que lo hacen funcionar. Math for Deep Learning le brindará conocimientos prácticos de probabilidad, estadística, álgebra lineal y cálculo diferencial, los subcampos matemáticos esenciales necesarios para practicar el aprendizaje profundo con éxito.

Cada subcampo se explica con código Python y ejemplos prácticos del mundo real que cierran la brecha entre las matemáticas puras y sus aplicaciones en el aprendizaje profundo. El libro comienza con fundamentos como el teorema de Bayes antes de avanzar a conceptos más avanzados como el entrenamiento de redes neuronales utilizando vectores, matrices y derivadas de funciones.

Luego, utilizará todos estos cálculos mientras explora e implementa la retropropagación y el descenso de gradiente, los algoritmos fundamentales que han permitido la revolución de la IA.

Aprenderás cómo:

- Utilizar estadísticas para comprender conjuntos de datos y evaluarlos.
- modelos

- Aplicar las reglas de probabilidad
- Manipular vectores y matrices para mover datos. a través de una red neuronal
- Usar álgebra lineal para implementar el principio análisis de componentes y valor singular descomposición
- Implementar técnicas de optimización basadas en gradientes. como RMSprop, Adagrad y Adadelta

Los conceptos matemáticos básicos presentados en Matemáticas para el aprendizaje profundo le brindarán la base que necesita para desbloquear el potencial del aprendizaje profundo en sus propias aplicaciones.

## SOBRE EL AUTOR

Ronald T. Kneusel ha estado trabajando con aprendizaje automático en la industria desde 2003. Obtuvo un doctorado en aprendizaje automático de la Universidad de Colorado, Boulder, en 2016. Kneusel también es autor de Practical Deep Learning (No Starch Press, 2021) y Números aleatorios y computadoras (Springer, 2018).

FUNDAS PITÓN 3.x



LO MEJOR EN ENTRETENIMIENTO GEEK™

[www.nostarch.com](http://www.nostarch.com)

\$49.99 (\$65.99 CDN)

ISBN 978-1-7185-0190-4  
  
 54999  
 9 781718 501904