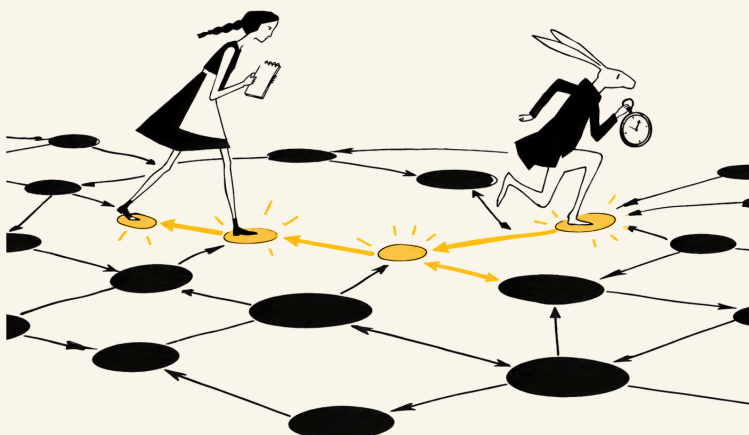


Las aventuras de Alicia en el País de las Maravillas Diferenciables



Simone Scardapane · 2024

Audiolecciones ilustradas

Lecturized

Las aventuras de Alicia en el País de las Maravillas Diferenciables

Audiolecciones ilustradas

Simone Scardapane · 2024

Contado como lecciones ilustradas por Lecturized

lecturized.com

Del original: Alice's Adventures in a Differentiable Wonderland, Volume I: A Tour of the
Land

Índice

Introducción

Sobre este libro

En el país de la diferenciabilidad

Notación y símbolos

Últimas reflexiones antes de partir

Licencia y preliminares matemáticos

Operaciones matriciales comunes

Complejidad computacional

La notación de Einstein y las derivadas

Gradientes, derivadas direccionales y jacobianos

Las dimensiones de los jacobianos y el descenso de gradiente

Convergencia del descenso de gradiente

Acelerar el descenso de gradiente

De la teoría a la práctica, y qué es un conjunto de datos

Variantes del aprendizaje supervisado

Funciones de pérdida

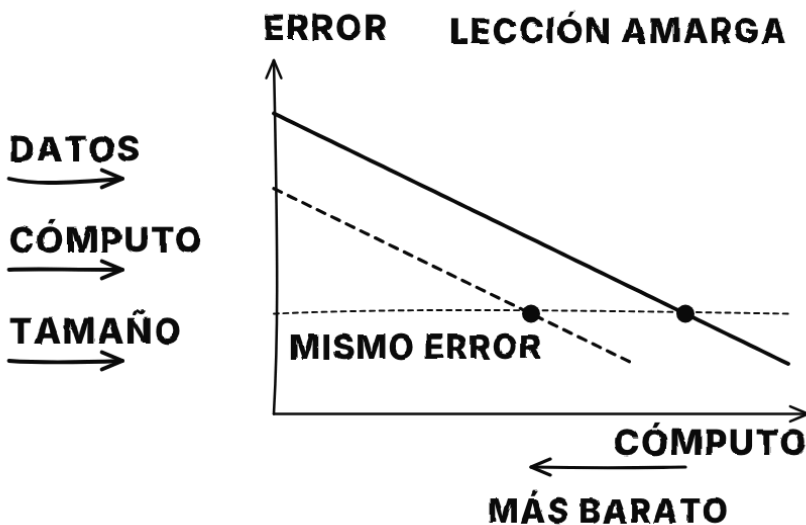
Riesgo esperado y elección de una pérdida

Máxima verosimilitud y aprendizaje bayesiano

Modelos lineales y la pérdida cuadrática

Introducción

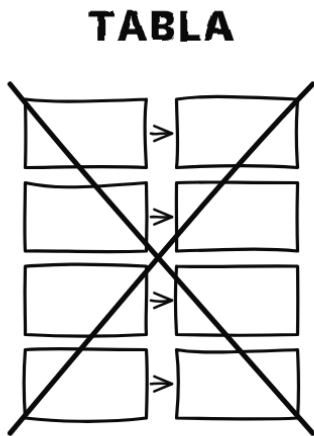
Las redes neuronales se han convertido en un componente integral de nuestro mundo cotidiano, ya sea abiertamente, bajo la forma de grandes modelos de lenguaje (LLM), o fuera de la vista, impulsando innumerables tecnologías y descubrimientos científicos, como drones, coches, motores de búsqueda, diseño molecular y sistemas de recomendación. Como veremos, todo esto se ha logrado apoyándose en un conjunto muy pequeño de principios rectores y componentes, que forman el núcleo de este libro, mientras que el foco de la investigación se ha desplazado a escalarlos hasta los límites de lo físicamente posible.



Más datos, cómputo, tamaño

El poder del escalado se encarna en el concepto relativamente reciente de leyes de escalado neuronal, que a su vez ha sido decisivo para impulsar inversiones masivas en inteligencia artificial (IA). Informalmente, para prácticamente cualquier tarea, aumentar a la vez los datos, la potencia de cómputo y el tamaño de los modelos casi siempre produce un aumento predecible de la precisión. Dicho de otro modo, la potencia de cómputo necesaria para alcanzar una precisión dada en una tarea disminuye en un factor constante por cada periodo de tiempo. Richard Sutton llamó la lección amarga al tremendo poder de combinar herramientas simples y de propósito general con una potencia de cómputo que crece exponencialmente en la IA.

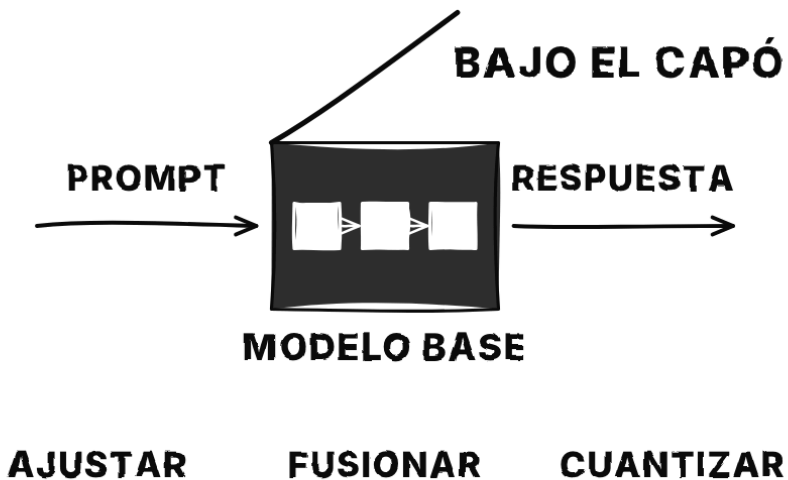
Si damos por sentadas las leyes de escalado, nos queda una herramienta casi mágica. En pocas palabras, las redes neuronales se optimizan para aproximar alguna distribución de probabilidad a partir de datos extraídos de ella. En principio, esta aproximación puede fallar: por ejemplo, las redes neuronales modernas son tan grandes que pueden memorizar fácilmente todos los datos que se les muestran y transformarse en una trivial tabla de consulta. En cambio, se ha mostrado que los modelos entrenados generalizan bien incluso a tareas que no se consideran explícitamente en los datos de entrenamiento. De hecho, a medida que crece el tamaño de los conjuntos de datos, el concepto de qué está dentro de la distribución y qué está fuera de ella se difumina, y los modelos a gran escala muestran indicios de fuertes capacidades de generalización y una fascinante baja dependencia de la memorización pura, es decir, del sobreajuste.



Memorizar o generalizar

La aparición de modelos extremadamente grandes que pueden aprovecharse para una variedad de tareas posteriores, a veces llamados modelos fundacionales, junto con una vibrante comunidad de código abierto, también ha cambiado la forma en que interactuamos con estos modelos. Muchas tareas pueden resolverse ahora simplemente mediante prompting, es decir, interactuando con instrucciones de texto o visuales con un modelo preentrenado encontrado en la web, mientras el interior del modelo sigue siendo una caja negra completa. Desde una perspectiva general, esto se parece al paso de tener que programar sus bibliotecas, por ejemplo en C++, a confiar en software de código abierto o comercial cuyo código fuente no es accesible. La metáfora no es tan descabellada como puede parecer: hoy en día, pocos equipos en el mundo tienen el

cómputo y la experiencia técnica para diseñar y publicar modelos verdaderamente a gran escala, como los LLM Llama, igual que pocas empresas tienen los recursos para construir software CRM empresarial.



Una caja negra, abierta

Y del mismo modo, igual que el software de código abierto ofrece infinitas posibilidades para personalizar o diseñar sus programas desde cero, el hardware de consumo y un poco de ingenio le dan un amplio abanico de opciones para experimentar con modelos diferenciados: desde ajustarlos a sus tareas hasta fusionar modelos, cuantizarlos para hardware de bajo consumo, probar su robustez o incluso diseñar variantes e ideas completamente nuevas. Para todo esto, necesita mirar «bajo el capó» y entender cómo estos modelos

procesan y manipulan los datos internamente, con todos sus trucos y peculiaridades nacidos de la experiencia y la depuración. Este libro es un punto de entrada a este mundo: si, como Alicia, usted es curioso por naturaleza, espero que aprecie el viaje.

Preguntas

1. ¿Qué dicen, informalmente, las leyes de escalado neuronal?
 - a. Aumentar a la vez los datos, el cómputo y el tamaño del modelo casi siempre da un aumento predecible de la precisión
 - b. Solo un mejor algoritmo de aprendizaje puede elevar la precisión una vez que un modelo es grande
 - c. Los modelos más grandes siempre memorizan sus datos y por eso empeoran en tareas nuevas
 - d. La precisión crece solo con el tamaño del modelo, sea cual sea la cantidad de datos
2. ¿Por qué podría esperarse que redes muy grandes fallaran al aproximar una distribución, y qué se observa en cambio?
 - a. Son demasiado lentas de entrenar, pero en la práctica se entrenan rápido
 - b. Podrían simplemente memorizar los datos como una tabla de consulta, pero los modelos entrenados generalizan bien incluso a tareas que no están en los datos de entrenamiento
 - c. Necesitan datos etiquetados, pero funcionan sin ningún dato
 - d. Olvidan los datos anteriores, pero conservan todo

3. ¿Qué ilustra la comparación con las bibliotecas en C++ y el software comercial cerrado?
 - a. Que las redes neuronales se escriben en C++
 - b. Que el prompting es más lento que programar
 - c. Que usar un modelo preentrenado mediante instrucciones es como confiar en software cuyo interior no puede ver, construido por los pocos que tienen los recursos
 - d. Que los modelos de código abierto no pueden modificarse
4. ¿Por qué dice el autor que necesita mirar bajo el capó de estos modelos?
 - a. Porque el prompting ya no funciona
 - b. Porque es ilegal usar modelos fundacionales sin inspeccionarlos
 - c. Porque solo así puede entrenarlos en un teléfono
 - d. Porque ajustar, fusionar, cuantizar, probar o diseñar nuevos modelos exige entender cómo procesan los datos internamente

Respuestas

1. **a)** El autor describe las leyes de escalado como un aumento predecible de la precisión cuando se aumentan a la vez los datos, la potencia de cómputo y el tamaño del modelo, para prácticamente cualquier tarea.
2. **b)** Las redes modernas son lo bastante grandes para memorizar todo lo que ven, pero el libro señala que se ha mostrado que los modelos entrenados, en cambio, generalizan bien.
3. **c)** Dar instrucciones a un modelo preentrenado lo trata como una caja negra, igual que se usa software comercial sin su código fuente; pocos

equipos pueden construir modelos verdaderamente grandes, como pocas empresas pueden construir software CRM empresarial.

4. d) Todas las opciones abiertas a quien experimenta, desde el ajuste fino hasta los diseños nuevos, exigen entender cómo los modelos manipulan los datos internamente, con sus trucos y peculiaridades.

Ejercicios

1. Con sus propias palabras, ¿a qué llamó Richard Sutton la lección amarga, y cómo se relaciona con las leyes de escalado?

Pista: Piense en qué se combina con qué.

Respuesta: La lección amarga es que herramientas simples y de propósito general, combinadas con una potencia de cómputo que crece exponencialmente, han sido tremendamente poderosas en la IA. Las leyes de escalado dicen lo mismo en forma medible: dar a tales modelos más datos, cómputo y tamaño mejora la precisión de forma predecible.

2. Explique por qué, a medida que crecen los conjuntos de datos, se difumina la línea entre lo que está dentro de la distribución y lo que está fuera.

Pista: Pregúntese qué fracción de las tareas posibles ya toca un conjunto de datos enorme.

Respuesta: Con conjuntos de datos muy grandes hay tantos tipos de ejemplos presentes que se vuelve difícil decir qué queda fuera de la distribución de entrenamiento. Los modelos grandes muestran entonces indicios de fuerte generalización y dependen poco de la memorización pura, es decir, del sobreajuste.

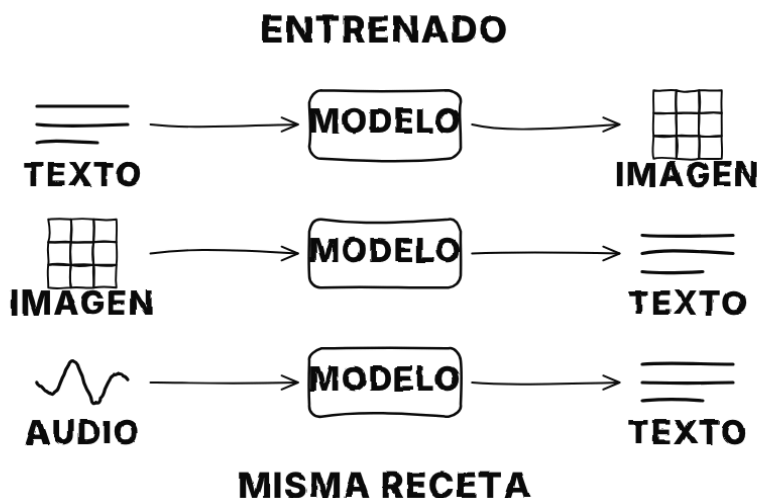
3. Enumere cuatro cosas que, según el autor, puede hacer con modelos diferenciables en hardware de consumo, y diga qué tienen en común.

Pista: Mire la lista que sigue a «desde ajustarlos».

Respuesta: Ajustarlos a sus tareas, fusionar modelos, cuantizarlos para hardware de bajo consumo y probar su robustez (o diseñar variantes nuevas). Todas exigen mirar bajo el capó y entender cómo los modelos procesan los datos internamente.

Sobre este libro

Suponemos que nuestros lectores conocen los fundamentos del aprendizaje automático (ML) y, más concretamente, del aprendizaje supervisado (SL). El aprendizaje supervisado puede usarse para resolver tareas complejas reuniendo datos sobre un comportamiento deseado y «entrenando», es decir, optimizando, sistemas para que aproximen ese comportamiento. Esta idea, engañosamente simple, es extremadamente poderosa. Por ejemplo, la generación de imágenes puede convertirse en el problema de reunir una colección suficientemente grande de imágenes con sus pies de foto; simular la lengua inglesa se convierte en la tarea de reunir una gran colección de textos y aprender a predecir una frase a partir de las anteriores; y diagnosticar una radiografía equivale a tener una gran base de datos de placas con las decisiones de los médicos asociadas, como muestra la figura F.1.2.

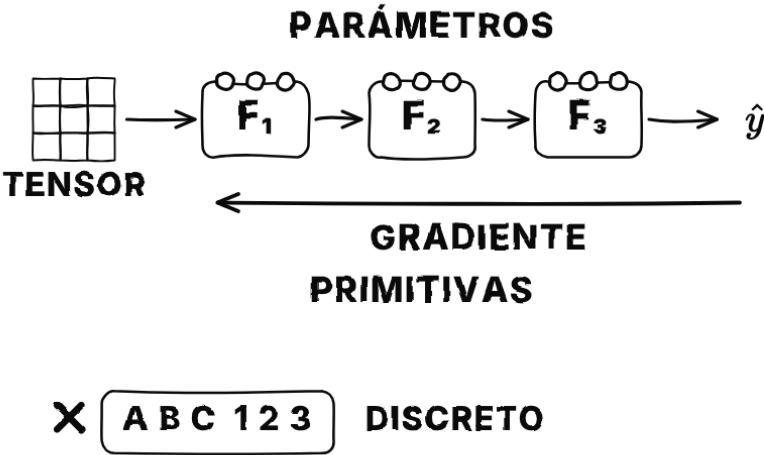


Tareas como entrada y salida

La figura ordena la mayoría de las tareas según la entrada y la salida que necesitan. La generación de imágenes quiere una imagen, una cuadrícula ordenada de píxeles, a partir de un texto, una secuencia de caracteres, mientras que la inversa, la descripción de imágenes, es el problema de generar un pie de foto a partir de una imagen. Como otro ejemplo, responder a una consulta de audio requiere un texto a partir de un audio, otra secuencia ordenada, esta vez numérica. Fascinantemente, el diseño de los modelos sigue especificaciones similares en todos los casos.

En general, aprender es un problema de búsqueda. Empezamos definiendo un programa con un gran número de grados de libertad, que llamamos parámetros, y manipulamos los parámetros hasta que

el rendimiento del modelo es satisfactorio. Para hacer práctica esta idea, necesitamos formas eficientes de buscar la configuración óptima incluso en presencia de millones, o miles de millones, o billones de parámetros. Como su nombre indica, los modelos diferenciables hacen esto restringiendo la elección del modelo a componentes diferenciables, es decir, funciones matemáticas que podemos derivar. Poder calcular una derivada de una función de alta dimensión, un gradiente, significa saber qué ocurre si perturbamos ligeramente sus parámetros, lo que a su vez conduce a rutinas automáticas para su optimización, sobre todo la diferenciación automática y el descenso de gradiente. Describir este planteamiento es el tema de la primera parte del libro, Brújula y aguja, que va del capítulo 2 al capítulo 6.



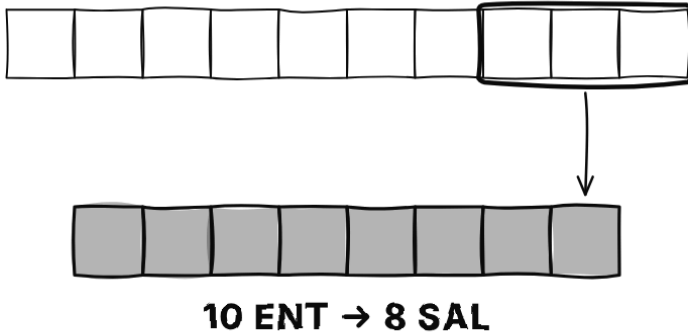
Una cadena de bloques diferenciables

Al ver las redes neuronales simplemente como composiciones de primitivas diferenciables, podemos plantear dos preguntas básicas, mostradas en la figura F.1.3. Primero, ¿qué tipos de datos podemos manejar como entradas o salidas? Y segundo, ¿qué clase de primitivas podemos usar? La diferenciabilidad es un requisito fuerte que no nos permite trabajar directamente con muchos tipos de datos estándar, como caracteres o enteros, que son fundamentalmente discretos y, por tanto, discontinuos. En cambio, veremos que los modelos diferenciables pueden trabajar fácilmente con datos más complejos representados como grandes matrices de números, que llamaremos tensores, como las imágenes, que pueden manipularse algebraicamente mediante composiciones básicas de transformaciones lineales y no lineales.

En la segunda parte del libro nos centramos en un ejemplo prototípico de componente diferenciable, el operador de convolución, del capítulo 7 al capítulo 9. Las convoluciones pueden aplicarse siempre que nuestros datos puedan representarse mediante una secuencia ordenada de elementos: entre otros, audio, imágenes, texto y vídeo. Por el camino también presentamos varias técnicas útiles para diseñar modelos profundos, es decir, modelos compuestos de muchos pasos en secuencia, así como varias ideas importantes como la tokenización de texto, la generación autorregresiva de secuencias y el modelado causal, que forman la base de los LLM más avanzados.

CONVOLUCIÓN TENSORES

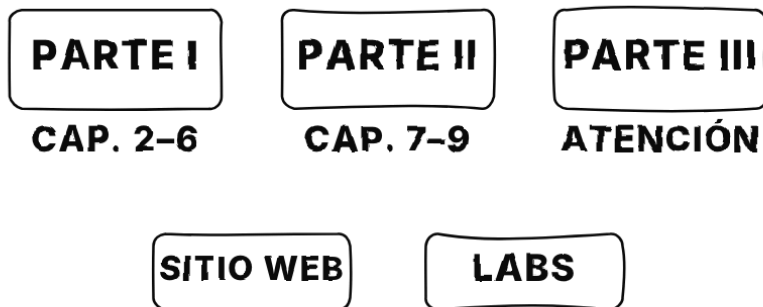
SECUENCIA



La convolución recorre una secuencia

La figura F.1.3 muestra las redes neuronales como secuencias de primitivas diferenciables que operan sobre matrices estructuradas, los tensores. Cada primitiva puede clasificarse según su firma de entrada y salida, que a su vez define las reglas para componerlas.

La tercera parte del libro, Por la madriguera del conejo, continúa nuestra exploración de los modelos diferenciables considerando diseños alternativos para conjuntos, sobre todo las capas de atención y los modelos transformadores en los capítulos 10 y 11, los grafos en el capítulo 12 y, por último, las capas recurrentes para secuencias temporales en el capítulo 13.



Las tres partes del libro

El libro se complementa con un sitio web donde, espero, reuniré capítulos y material adicionales sobre temas de interés que no se centran en un tipo concreto de datos, como el modelado generativo, la computación condicional, el aprendizaje por transferencia y la explicabilidad. Estos capítulos tienen un carácter más orientado a la investigación y pueden leerse en cualquier orden. Además, ofrezco una serie de sesiones de laboratorio guiadas en forma de cuadernos, que cubren gran parte del material del libro así como temas avanzados como el aprendizaje contrastivo y la fusión de modelos.

Preguntas

1. ¿Por qué el libro llama al aprendizaje un problema de búsqueda?
 - a. Porque los modelos buscan las respuestas en la web
 - b. Porque definimos un programa con muchos parámetros y los ajustamos hasta que el rendimiento es satisfactorio
 - c. Porque hay que buscar errores en los datos de entrenamiento
 - d. Porque cada tarea tiene exactamente un programa correcto
2. ¿Qué nos da conocer el gradiente de una función de alta dimensión?
 - a. Los parámetros óptimos exactos en un solo paso
 - b. Una lista de todos los datos de entrenamiento
 - c. Saber qué ocurre cuando los parámetros se perturban ligeramente, lo que conduce a la optimización automática
 - d. Una forma de trabajar directamente con caracteres y enteros
3. ¿Por qué los modelos diferenciables no pueden trabajar directamente con caracteres o enteros?
 - a. Esos tipos de datos son discretos y, por tanto, discontinuos, mientras que la diferenciabilidad necesita continuidad
 - b. Esos tipos de datos ocupan demasiada memoria
 - c. Los caracteres y los enteros no pueden guardarse en tensores en absoluto
 - d. Sí pueden, pero solo con convoluciones
4. ¿Cuándo pueden aplicarse las convoluciones?
 - a. Solo a imágenes
 - b. Solo a datos sin orden

- c. Siempre que los datos puedan representarse como una secuencia ordenada de elementos, como audio, imágenes, texto y vídeo
- d. Solo después de convertir los datos en enteros

Respuestas

- 1. **b)** El aprendizaje parte de un programa con un gran número de grados de libertad, los parámetros, que se manipulan hasta que el modelo rinde bien.
- 2. **c)** Un gradiente dice qué ocurre ante una ligera perturbación de los parámetros, y eso conduce a rutinas como la diferenciación automática y el descenso de gradiente.
- 3. **a)** La diferenciabilidad es un requisito fuerte; los caracteres y los enteros son fundamentalmente discretos y, por tanto, discontinuos.
- 4. **c)** El libro dice que las convoluciones se aplican siempre que los datos sean una secuencia ordenada de elementos, y enumera audio, imágenes, texto y vídeo.

Ejercicios

- 1. Elija uno de los ejemplos del libro (generación de imágenes, simulación del lenguaje, diagnóstico de radiografías) y explique cómo se convierte en un problema de aprendizaje supervisado.

Pista: Diga qué datos reuniría y qué comportamiento entrenaría.

Respuesta: Para el diagnóstico de radiografías, se reúne una gran base de datos de placas con las decisiones de los médicos adjuntas y se entrena un sistema para que aproxime ese comportamiento: dada una placa, producir la decisión. La tarea se convierte en reunir ejemplos del comportamiento entrada-salida deseado y optimizar un modelo para que lo imite.

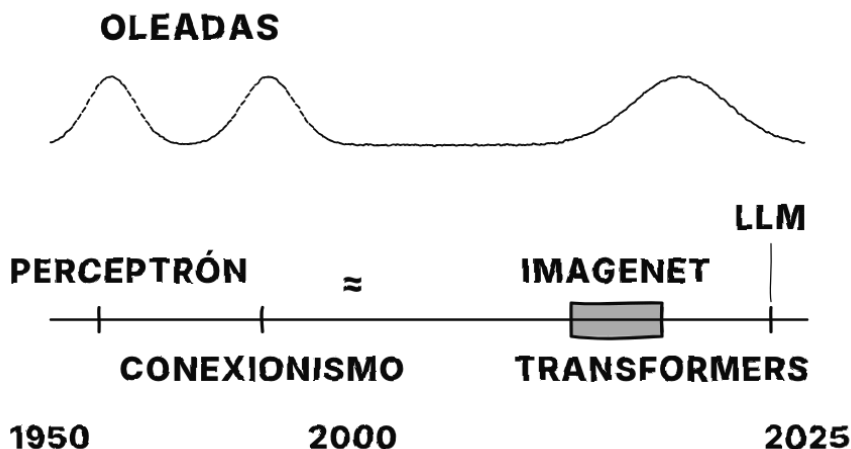
2. Nombre las dos preguntas básicas que plantea el libro una vez que las redes neuronales se ven como composiciones de primitivas diferenciables, y diga qué partes del libro las responden para qué datos.

Pista: Una pregunta trata de los tipos de datos, la otra de los bloques de construcción.

Respuesta: ¿Qué tipos de datos podemos manejar como entradas o salidas, y qué primitivas podemos usar? La primera parte establece el marco diferenciable, la segunda estudia las convoluciones para secuencias ordenadas como audio, imágenes, texto y vídeo, y la tercera cubre los conjuntos (atención y transformadores), los grafos y las secuencias temporales (capas recurrentes).

En el país de la diferenciabilidad

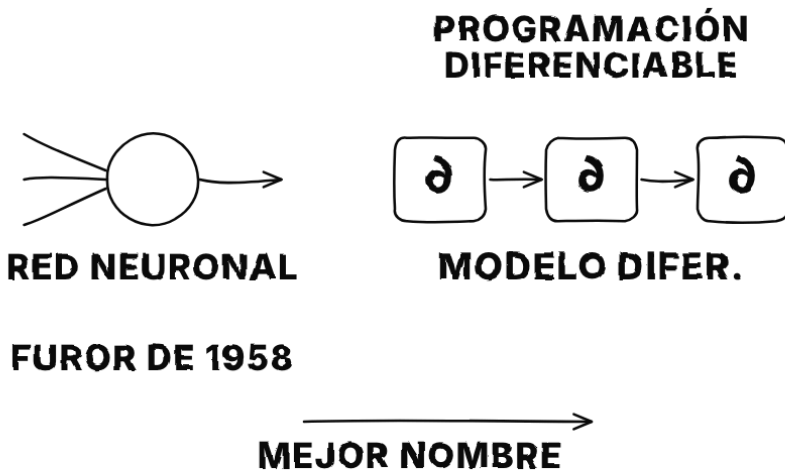
Las redes neuronales tienen una historia larga y rica. El propio nombre es un eco de los primeros intentos de modelar neuronas biológicas en el siglo veinte, y una terminología parecida ha seguido siendo omnipresente: para ser coherentes con los marcos existentes, en los próximos capítulos podemos hablar de neuronas, capas o, por ejemplo, activaciones. Tras varias olas de interés, el periodo entre 2012 y 2017 vio un aumento sin precedentes de la complejidad de las redes, espoleado por pruebas de referencia y competiciones a gran escala, sobre todo el ImageNet Large Scale Visual Recognition Challenge (ILSVRC), que tratamos en el capítulo 9. Una segunda gran ola de interés vino de la introducción de los transformadores, el tema del capítulo 10, en 2017: igual que la visión por computador había sido conquistada por los modelos convolucionales unos años antes, el procesamiento del lenguaje natural fue conquistado por los transformadores en muy poco tiempo. En estos años se lograron mejoras adicionales para vídeos, grafos, que son el tema del capítulo 12, y audio, culminando en el entusiasmo actual en torno a los LLM, las redes multimodales y los modelos generativos. Este no es el lugar para un repaso histórico completo de las redes neuronales modernas.



Olas de redes neuronales

Este periodo fue paralelo a una rápida evolución de la terminología, desde el conexionismo de los ochenta hasta el uso de aprendizaje profundo para referirse a las redes modernas, en oposición a los modelos más pequeños y menos profundos del pasado. A pesar de ello, todos estos términos siguen siendo inexorablemente vagos, porque las redes artificiales modernas casi no conservan ningún parecido con las redes neuronales biológicas ni con la neurología. Mirando las redes neuronales modernas, su característica esencial es estar compuestas de bloques diferenciables: por esta razón, en este libro prefiero el término modelos diferenciables cuando es posible. Ver las redes neuronales como modelos diferenciables conduce directamente al tema más amplio de la programación diferenciable, una disciplina emergente que mezcla la informática y la optimización

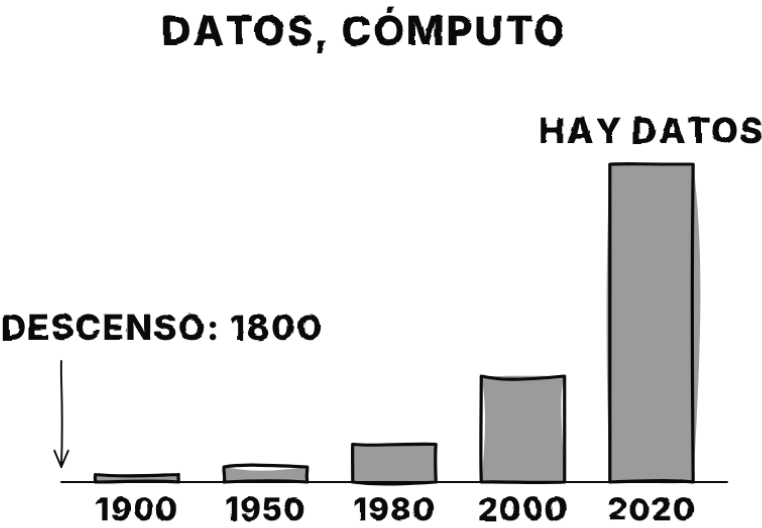
para estudiar los programas informáticos diferenciables de forma más general. Como muchos, me inspiró un «manifiesto» publicado por Yann LeCun en Facebook en 2018, y por la conexión entre las redes neuronales y la programación y el desarrollo de código abierto también agradezco un segundo manifiesto, publicado por Colin Raffel en 2021.



No neuronas: bloques diferenciables

Una figura del libro muestra el bombo en torno a la IA, salvo que es 1958, y el psicólogo estadounidense Frank Rosenblatt ha atraído una notable atención de los medios con sus estudios sobre «perceptrones», uno de los primeros prototipos funcionales de redes neuronales.

Mientras viajamos por este país de los modelos diferenciables, viajamos también por la historia. Los conceptos básicos de la optimización numérica de modelos lineales por descenso de gradiente, tratados en el capítulo 4, se conocían al menos desde el siglo diecinueve; las llamadas «redes totalmente conectadas», en la forma en que las usamos más adelante, pueden datarse en los años ochenta; los modelos convolucionales ya se conocían y se usaban a finales de los noventa. Para una historia extensa y con opinión, existe también una historia anotada de las redes neuronales de Jürgen Schmidhuber. Sin embargo, hicieron falta muchas décadas para disponer de suficientes datos y potencia y darse cuenta de lo bien que pueden rendir estos modelos con suficientes datos y suficientes parámetros.



Ideas viejas, escala nueva

Aunque no tenemos espacio para profundizar en todos los temas posibles, también por lo rápido que avanza la investigación, espero que el libro ofrezca material suficiente para que el lector pueda orientarse con facilidad en la literatura más reciente.

Preguntas

1. ¿Por qué el autor prefiere el término modelos diferenciables al de redes neuronales?
 - a. Porque las redes modernas son biológicamente exactas
 - b. Porque su característica esencial es estar compuestas de bloques diferenciables, y casi no conservan parecido con las neuronas biológicas
 - c. Porque redes neuronales es una marca registrada
 - d. Porque los modelos diferenciables son más pequeños que las redes neuronales
2. ¿Qué pasó con el procesamiento del lenguaje natural tras la aparición de los transformadores en 2017?
 - a. Fue conquistado por los transformadores en muy poco tiempo, como la visión por computador lo había sido por los modelos convolucionales
 - b. Volvió a los sistemas basados en reglas
 - c. Dejó de usar redes neuronales
 - d. Se fusionó con ImageNet
3. ¿Por qué ideas conocidas desde el siglo diecinueve o los años ochenta tardaron tanto en mostrar su poder?

- a. Al principio eran matemáticamente erróneas
- b. Sus inventores las mantuvieron en secreto
- c. Hicieron falta décadas para disponer de suficientes datos y potencia de cómputo y ver lo bien que rinden a gran escala
- d. Solo funcionan con transformadores

Respuestas

- 1. **b)** Los términos en torno a las neuronas son vagos, ya que las redes modernas apenas se parecen a la biología; lo que las define es que están construidas con bloques diferenciables.
- 2. **a)** El libro traza el paralelo: la visión por computador fue conquistada por los modelos convolucionales, y unos años después el procesamiento del lenguaje por los transformadores.
- 3. **c)** El descenso de gradiente, las redes totalmente conectadas y las convoluciones eran todas ideas viejas; lo que faltaba eran suficientes datos y potencia.

Ejercicios

- 1. Coloque esto en una línea de tiempo usando solo lo que dice el pasaje: el descenso de gradiente para modelos lineales, las redes totalmente conectadas, los modelos convolucionales, el bombo del perceptrón de Rosenblatt, el auge impulsado por ImageNet, los transformadores.

Pista: El pasaje da un siglo, una década o un año para cada uno.

Respuesta: Descenso de gradiente para modelos lineales: al menos el siglo diecinueve. Bombo del perceptrón: 1958. Redes totalmente conectadas: años ochenta. Modelos convolucionales: finales de los noventa. Auge impulsado por ImageNet: de 2012 a 2017. Transformadores: 2017.

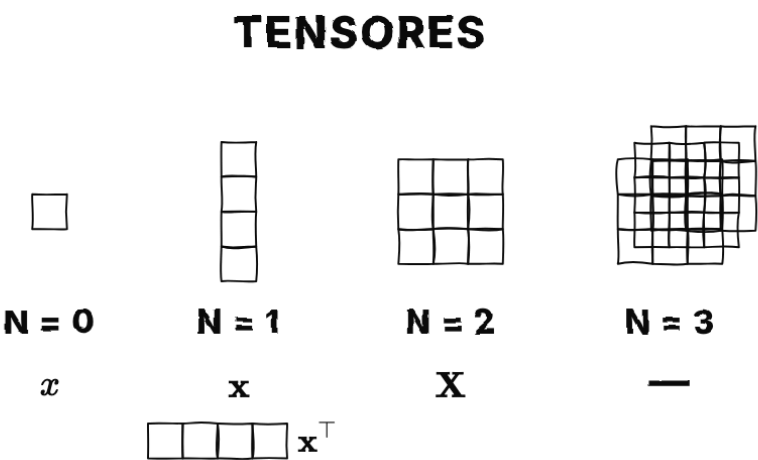
2. Con sus propias palabras, ¿qué es la programación diferenciable, según el pasaje?

Pista: Mezcla dos campos.

Respuesta: Es una disciplina emergente que mezcla la informática y la optimización para estudiar los programas informáticos diferenciables en general; ver las redes neuronales como modelos diferenciables conduce directamente a ella.

Notación y símbolos

Una figura del libro muestra los tipos de datos fundamentales: escalares, vectores, matrices y matrices genéricas de n dimensiones. Usamos el nombre tensores para referirnos a ellos, y n se llama el rango del tensor. La figura muestra el vector como una fila para facilitar la lectura, pero en el texto suponemos que todos los vectores son vectores columna.



Escalar, vector, matriz, tensor

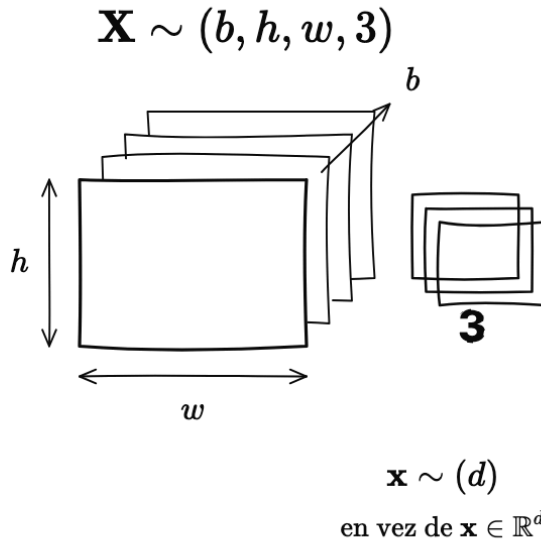
El tipo de dato fundamental al trabajar con modelos diferenciables es un tensor, que definimos como una matriz de n dimensiones de

objetos, normalmente números reales. En la literatura científica, los tensores tienen una definición más precisa como operadores multilineales, mientras que los objetos que usamos en el libro son simples matrices multidimensionales; aunque es un nombre inexacto, el uso de tensor está tan extendido que mantenemos aquí esta convención. Con disculpas a cualquier matemático que nos lea, llamamos a n el rango del tensor. La notación del libro varía según n .

Un tensor de un solo elemento, con $n = 0$, es simplemente un único valor, un escalar. Para los escalares usamos letras minúsculas, como x o y . Si se lo pregunta, los escalares se llaman así porque pueden escribirse como múltiplos escalares de uno; además, prometo reducir el número de notas al pie a partir de ahora. Las columnas de valores, con $n = 1$, son vectores. Para los vectores usamos una fuente minúscula en negrita, como $\mathbf{x}$. El vector fila correspondiente se denota por $\mathbf{x}^T$ cuando necesitamos distinguirlos, y también podemos omitir la traspuesta para facilitar la lectura, si queda claro por el contexto. Las matrices rectangulares de valores, con $n = 2$, son matrices. Usamos una fuente mayúscula en negrita, como $\mathbf{X}$ o $\mathbf{Y}$. No se usa ninguna notación específica para $n > 2$. Evitamos los símbolos caligráficos como $\mathcal{X}$, que reservamos para conjuntos o distribuciones de probabilidad.

Para trabajar con tensores usamos diversas estrategias de indexación, descritas mejor en la sección 2.1. En la mayoría de los casos, entender un algoritmo o una operación se reduce a entender la forma de cada tensor implicado. Para denotar la forma de manera concisa, usamos la siguiente notación: $\mathbf{X} \sim (b, h, w, 3)$. Este es un tensor de rango 4 con forma $(b, h, w, 3)$. Algunas dimensiones pueden estar fijadas de antemano, como el 3, mientras que otras dimensiones pueden denotarse con variables. Usamos el mismo símbolo para denotar la

extracción de una distribución de probabilidad, por ejemplo $\varepsilon \sim \mathcal{N}(0, 1)$, pero lo hacemos raramente, y el significado del símbolo siempre debe quedar claro por el contexto. Por tanto, $\mathbf{x} \sim (d)$ sustituirá al más común $\mathbf{x} \in \mathbb{R}^d$, y de modo similar $\mathbf{X} \sim (n, d)$ en lugar de $\mathbf{X} \in \mathbb{R}^{n \times d}$.



Formas: $\mathbf{X} \sim (b, h, w, 3)$

Por último, puede que queramos restringir los elementos de un tensor, para lo cual usamos una notación especial. $\mathbf{x} \sim \text{Binary}(c)$ denota un tensor con solo valores binarios, es decir, elementos que son 0 o 1. $\mathbf{x} \sim \Delta(a)$ denota un vector que pertenece al llamado *símplex*, es decir, $x_i \geq 0$ y $\sum_i x_i = 1$. Para tensores de rango mayor, por ejemplo $\mathbf{X} \sim \Delta(n, c)$, suponemos que la normalización se aplica respecto a la última dimensión; en este caso, cada fila $\mathbf{X}_i$ pertenece al *símplex*.

Binary(5)

0	1	1	0	1
----------	----------	----------	----------	----------

$$\mathbf{X} \sim \Delta(2, 3)$$

FILAS	→	0.5	0.3	0.2	= 1
	→	0.1	0.6	0.3	= 1

Binario y el s mplex

En cada cap tulo se introduce notaci n adicional cuando es necesario. Tamb n tenemos unos pocos s mbolos al margen.

Preguntas

1.  Cu l es el rango de una matriz en la notaci n del libro?
 - a. 0
 - b. 1
 - c. 2
 - d. Depende del n mero de filas

2. ¿Qué le dice $X \sim (b, h, w, 3)$?
- a. Que X se extrae de una distribución con cuatro parámetros
 - b. Que X es un tensor de rango 4 cuya forma es b por h por w por 3
 - c. Que X tiene cuatro elementos
 - d. Que X es una matriz con b filas
3. ¿Qué vector pertenece al simplex $\Delta(3)$?
- a. $(0.6, 0.6, -0.2)$
 - b. $(0.5, 0.5, 0.5)$
 - c. $(1, 1, 1)$
 - d. $(0.2, 0.5, 0.3)$
4. ¿Cómo escribe el libro un vector fila, dado que sus vectores son columnas?
- a. Con una traspuesta, $\mathbf{x}^T$
 - b. Con una letra mayúscula en negrita
 - c. Con una letra caligráfica
 - d. Con una tilde

Respuestas

- 1. c)** Las matrices rectangulares de valores tienen $n = 2$, y n es lo que el libro llama el rango.
- 2. b)** La tilde seguida de una forma denota la forma del tensor; aquí cuatro dimensiones, así que rango 4, con la última fijada en 3.
- 3. d)** Un vector del simplex tiene entradas no negativas que suman uno; solo $(0.2, 0.5, 0.3)$ cumple ambas condiciones.

4. a) Los vectores son columnas, y el vector fila correspondiente se denota $\mathbf{x}^T$ cuando hay que distinguir los dos.

Ejemplos resueltos

Forma de un lote de fotos Un lote contiene 32 fotos en color, cada una de 64 píxeles de alto y 64 de ancho, con 3 canales de color.

Escriba su forma en la notación del libro, dé su rango y el número de valores que contiene.

El libro escribe las formas con la tilde: enumere las dimensiones en orden, primero el lote, luego alto, ancho y canales.

Así, $\mathbf{X} \sim (32, 64, 64, 3)$.

El rango es el número de dimensiones, aquí cuatro.

El número de valores es el producto de las dimensiones: $32 \times 64 \times 64 \times 3 = 32 \times 4096 \times 3 = 393\,216$.

Respuesta: $\mathbf{X} \sim (32, 64, 64, 3)$, un tensor de rango 4 que contiene 393 216 valores.

Ejercicios

1. ¿Está el vector $(0.6, 0.6, -0.2)$ en el simplex $\Delta(3)$? Explique.

Pista: Compruebe las dos condiciones del simplex, no solo la suma.

Respuesta: No. Sus entradas suman 1, pero una entrada es negativa, y el simplex exige que todo $x_i \geq 0$.

2. Una tabla de 100 clientes con 8 características cada uno se escribe $\mathbf{X} \sim (100, 8)$. ¿Cómo quedaría la misma afirmación en la notación más común, y cuál es su rango?

Pista: El libro dice qué notación sustituye su tilde.

Respuesta: $\mathbf{X} \in \mathbb{R}^{100 \times 8}$, una matriz, así que un tensor de rango 2.

3. Si $\mathbf{Y} \sim \Delta(5, 4)$, ¿qué debe cumplirse para cada fila de $\mathbf{Y}$?

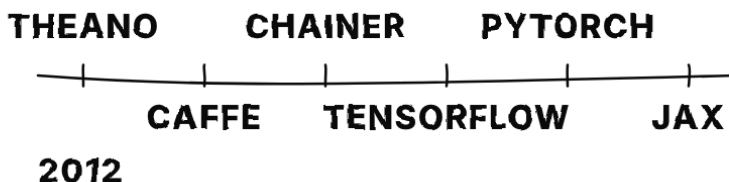
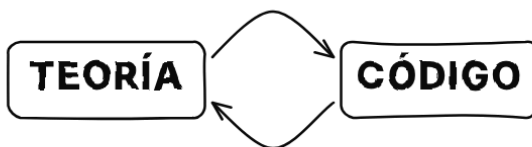
Pista: Para rangos mayores la normalización se aplica a la última dimensión.

Respuesta: Cada una de las 5 filas tiene 4 entradas no negativas que suman 1, así que cada fila pertenece al simplex $\Delta(4)$.

Últimas reflexiones antes de partir

El libro nace de mi deseo de dar una forma coherente a mis clases de Redes Neuronales para Aplicaciones de Ciencia de Datos, un curso que he impartido durante muchos años en el máster en Ciencia de Datos de la Universidad Sapienza de Roma. Los capítulos centrales del libro constituyen la parte principal del curso, mientras que los capítulos restantes son temas que trato de forma intermitente según el año. Algunas partes se han complementado con otros cursos que he impartido, o que pienso impartir, incluidas partes de Redes Neuronales para Ingeniería Informática, una introducción al aprendizaje automático para Ingeniería de Telecomunicaciones, además de algunos tutoriales, cursos de doctorado y escuelas de verano a lo largo de los años.

CURSO SAPIENZA



Teoría y código, de la mano

Ya existen varios libros excelentes y recientes sobre el tema de las redes neuronales profundas modernas. Este libro cubre al principio un contenido similar al de todos ellos, mientras que la exposición y algunas partes adicionales, o unas pocas secciones de los capítulos avanzados, se cruzan menos con ellos y dependen sobre todo de mis intereses de investigación. Espero poder ofrecer un punto de vista adicional y complementario sobre el material existente.

Como sugiere mi elección del nombre, entender los programas diferenciables viene tanto de la teoría como de la programación: hay una interacción constante entre cómo diseñamos los modelos y cómo los implementamos, y temas como la diferenciación automática son el mejor ejemplo. El resurgimiento actual de las redes neuronales,

aproximadamente desde 2012, puede atribuirse en gran parte a la disponibilidad de potentes bibliotecas de software, desde Theano hasta Caffe, Chainer y luego directamente las versiones modernas de TensorFlow, PyTorch y JAX, entre otras. Siempre que es posible, intento conectar la discusión con conceptos de los marcos de programación existentes, con énfasis en PyTorch y JAX. Sin embargo, el libro no es un manual de programación, y remito a la documentación de las bibliotecas para una introducción completa a cada una de ellas.

Antes de seguir, me gustaría enumerar algunas cosas más que este libro no es. Primero, he intentado escoger unos pocos conceptos que son a la vez comunes hoy y lo bastante generales para ser útiles en el futuro cercano. Sin embargo, no puedo prever el futuro y no aspiro a la exhaustividad, y varias partes de estos capítulos pueden estar incompletas o desactualizadas cuando usted las lea. Segundo, para cada concepto intento ofrecer algunos ejemplos de variantes que existen en la literatura, por ejemplo de la normalización por lotes a la normalización por capas. Sin embargo, tenga en cuenta que existen cientos más: para ello, le invito a explorar las muchas páginas de Papers With Code. Por último, este es un libro sobre los componentes fundamentales de los modelos diferenciables, pero implementarlos a escala, y hacer que funcionen, requiere tanto sofisticación de ingeniería como un poco de intuición. Trato poco el lado del hardware, y para lo segundo nada supera la experiencia y las entradas de blog con opinión, como la receta de entrenamiento de Andrej Karpathy o su serie de vídeos Zero to Hero.

Agradecimientos. El coloreado de las ecuaciones se debe a un bello paquete de LaTeX de S. T. John. Las imágenes en color de Alicia en el país de las maravillas y los símbolos en blanco y negro del margen

tienen licencia de Shutterstock.com. Las imágenes de Alicia en el país de las maravillas en las figuras del texto principal son reproducciones de las ilustraciones originales de John Tenniel, gracias a Wikimedia. Agradezco a Roberto Alma sus comentarios sobre un borrador anterior del libro y que me animara a publicarlo. También agradezco a Corrado Zoccolo, Emanuele Rodolà, Marcin Ślaby, Konstantin Burlachenko y Diego Sandoval sus amplias correcciones y sugerencias a la versión actual, y a todos los que me enviaron comentarios por correo electrónico.

Preguntas

1. ¿Qué da el autor como razón principal del resurgimiento de las redes neuronales desde aproximadamente 2012?
 - a. El descubrimiento del perceptrón
 - b. La disponibilidad de potentes bibliotecas de software, desde Theano y Caffe hasta TensorFlow, PyTorch y JAX
 - c. El fin del desafío ImageNet
 - d. Una nueva teoría de las neuronas biológicas
2. ¿Por qué el libro conecta constantemente las ideas con PyTorch y JAX sin ser un manual de programación?
 - a. Porque hay una interacción constante entre cómo se diseñan los modelos y cómo se implementan
 - b. Porque se espera que el lector no sepa nada de matemáticas
 - c. Porque las bibliotecas sustituyen a la teoría
 - d. Porque el autor escribió esas bibliotecas

3. ¿Cuál de estas cosas dice el autor que el libro no es?
- a. Un libro sobre los componentes fundamentales de los modelos diferenciables
 - b. Un catálogo completo y a prueba de futuro de cada variante y del lado del hardware
 - c. Un libro basado en un curso universitario
 - d. Un libro que conecta con los marcos existentes

Respuestas

1. **b)** El resurgimiento puede atribuirse en gran parte a potentes bibliotecas de software, y por eso el libro conecta la teoría con el código.
2. **a)** Entender los programas diferenciables viene de la teoría y la programación juntas; la diferenciación automática es el mejor ejemplo de esa interacción.
3. **b)** El autor no aspira a la exhaustividad, espera que algunas partes envejeczan, remite a cientos de variantes en otros lugares y trata poco el hardware.

Ejercicios

1. Enumere con sus propias palabras las tres cosas que el autor dice que este libro no es.

Pista: Empiezan con Primero, Segundo y Por último.

Respuesta: No es completo ni está garantizado que esté al día, ya que el campo avanza y algunas partes pueden quedar desactualizadas; no es un catálogo de cada variante, solo unos pocos ejemplos entre cientos; y no es una guía para implementar modelos a escala ni sobre hardware, que requieren habilidad de ingeniería, intuición y experiencia.

2. ¿Por qué la diferenciación automática es el mejor ejemplo del autor de la interacción entre diseño e implementación?

Pista: Piense en qué tiene que existir en el software para que un modelo pueda entrenarse.

Respuesta: Diseñar modelos diferenciables solo compensa si el software puede calcular los gradientes de forma eficiente; la diferenciación automática es el algoritmo que los marcos implementan para hacerlo, así que el diseño del modelo y su implementación están ligados ahí con la mayor claridad.

Licencia y preliminares matemáticos

El libro se publica bajo la licencia CC BY-SA. Esta licencia permite «a los reutilizadores distribuir, remezclar, adaptar y construir sobre el material en cualquier medio o formato, siempre que se dé crédito al creador. La licencia permite el uso comercial. Si remezcla, adapta o construye sobre el material, debe licenciar el material modificado en términos idénticos».

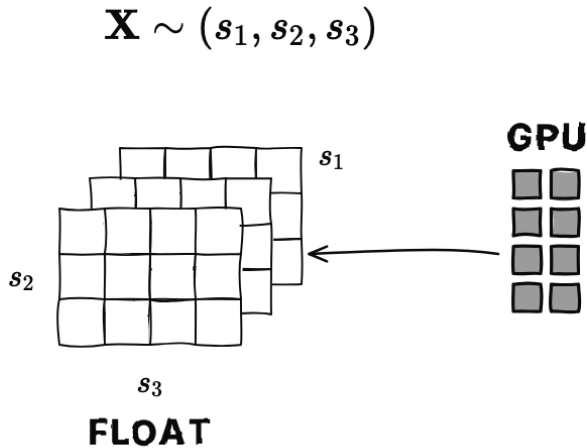
Capítulo 2, Preliminares matemáticos. En resumen: comprimimos aquí los conceptos matemáticos necesarios para seguir el libro. Suponemos un conocimiento previo de todos estos temas, y nos centramos más en describir la notación concreta y dar una visión general cohesionada. Cuando es posible, subrayamos la relación entre parte de este material, por ejemplo los tensores, y su implementación en la práctica.

El capítulo se compone de tres partes que se siguen una de otra: empezando por el álgebra lineal, pasando a la definición de gradientes para objetos de n dimensiones y, por último, a cómo podemos optimizar funciones aprovechando tales gradientes. En un apéndice se da una visión general autocontenida de la teoría de la probabilidad, con énfasis en el principio de máxima verosimilitud.

Este capítulo está lleno de contenido y definiciones: itenga paciencia conmigo un rato!

Sección 2.1, Álgebra lineal. Recordamos aquí algunos conceptos básicos de álgebra lineal que serán útiles más adelante y que nos

permiten acordar una notación compartida. La mayor parte del libro gira en torno a la idea de tensor.



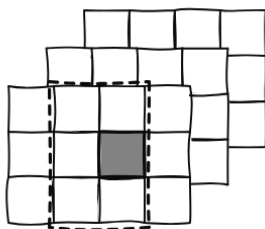
Un tensor y su forma

Definición, tensores: un tensor $\mathbf{X}$ es una matriz de n dimensiones de elementos del mismo tipo. En el libro usamos $\mathbf{X} \sim (s_1, s_2, \dots, s_n)$ para denotar rápidamente la forma del tensor.

Para $n = 0$ obtenemos escalares, valores únicos, mientras que tenemos vectores para $n = 1$, matrices para $n = 2$ y matrices de mayor dimensión en los demás casos. Recuerde que usamos x minúscula para los escalares, $\mathbf{x}$ minúscula en negrita para los vectores y $\mathbf{X}$ mayúscula en negrita para las matrices. Los tensores en el sentido descrito aquí son fundamentales en el aprendizaje profundo porque se

adaptan bien a una implementación masivamente paralela, como el uso de GPU o hardware más especializado, por ejemplo TPU e IPU.

Un tensor se describe por el tipo de sus elementos y su forma. La mayor parte de nuestra discusión se centrará en tensores de valores de coma flotante, cuyo formato concreto consideraremos más adelante, pero también pueden definirse para enteros, por ejemplo en clasificación, o para cadenas, por ejemplo para texto. Los tensores pueden indexarse para obtener rebanadas, es decir, subconjuntos de sus valores, y se aplican la mayoría de las convenciones de indexación de NumPy; para facilitar la lectura, en el libro indexamos desde 1, no desde 0. Para ecuaciones simples usamos subíndices: por ejemplo, para un tensor tridimensional $X \sim (a, b, c)$ podemos escribir X_i para denotar una rebanada de tamaño (b, c) , o $X_{i j k}$ para un único escalar. Usamos comas para expresiones más complejas, como $X_{i,:j:k}$ para denotar una rebanada de tamaño $(b, k - j)$. Cuando es necesario para evitar el desorden, usamos una notación en gris claro, $[X]_{i j k}$, para separar visualmente la parte de indexación del resto, donde el argumento dentro de los corchetes también puede ser una expresión.



$$\mathbf{X}_i \sim (b, c)$$

$$X_{ijk}$$

$$\mathbf{X}_{i,:j:k}$$

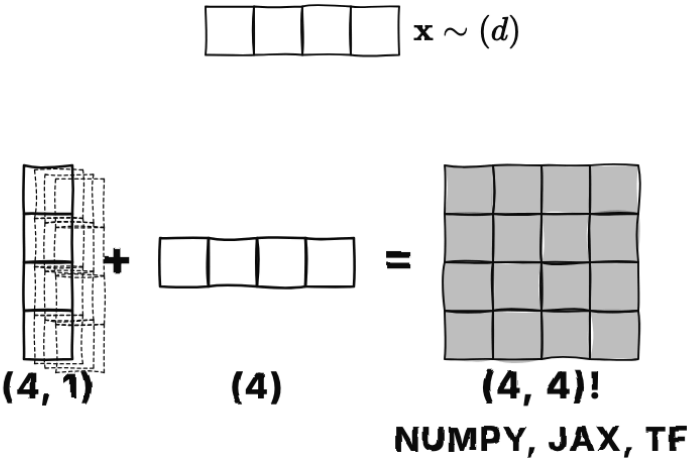
+ × EXP SIN √

Rebanar un tensor

Operaciones vectoriales comunes. Nos ocupamos sobre todo de modelos que pueden escribirse como composiciones de operaciones diferenciables. De hecho, la mayoría de nuestros modelos consistirán en composiciones básicas de sumas, multiplicaciones y algunas no linealidades adicionales como la exponencial $\exp(x)$, senos y cosenos, y raíces cuadradas.

Los vectores $\mathbf{x} \sim (d)$ son ejemplos de tensores unidimensionales. Los libros de álgebra lineal se preocupan por distinguir entre vectores columna $\mathbf{x}$ y vectores fila $\mathbf{x}^T$, y trataremos de ceñirnos a esta convención tanto como sea posible. En código esto es más delicado, porque los vectores fila y columna corresponden a tensores bidimensionales de forma $(1, d)$ o $(d, 1)$, que son distintos de los

tensores unidimensionales de forma (d). Es importante tenerlo en cuenta porque la mayoría de los marcos implementan reglas de difusión inspiradas en NumPy, lo que da lugar a comportamientos poco intuitivos; en pocas palabras, la difusión alinea las formas de los tensores desde la derecha y repite un tensor siempre que es posible para igualar las dos formas. El recuadro C.2.1 muestra un ejemplo de un error muy común que surge de la difusión implícita de las formas de los tensores.

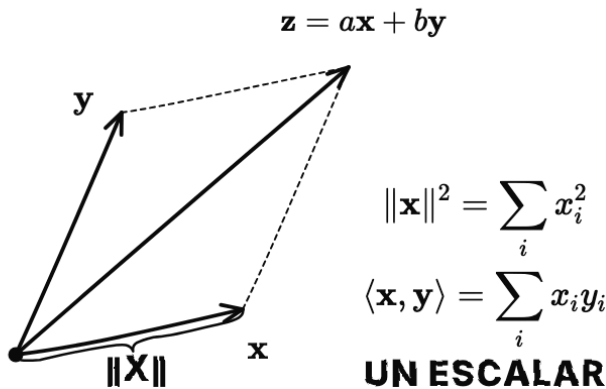


Una trampa de difusión

En ese recuadro, que probablemente es una difusión incorrecta, una «columna» de forma 4 por 1 se suma a un tensor unidimensional de forma 4, y el resultado tiene forma 4 por 4: una operación elemento a elemento sobre dos vectores produce una matriz a causa de sus

formas. El mismo resultado puede obtenerse en prácticamente cualquier marco, como NumPy, TensorFlow o JAX.

Los vectores poseen su propia álgebra, que llamamos espacio vectorial, en el sentido de que dos vectores cualesquiera $\mathbf{x}$ e $\mathbf{y}$ de la misma forma pueden combinarse linealmente, $\mathbf{z} = a\mathbf{x} + b\mathbf{y}$, para dar un tercer vector, con $z_i = ax_i + by_i$.



Sumar vectores, medir longitud

Si entendemos un vector como un punto en el espacio euclidiano de d dimensiones, la suma se interpreta formando un paralelogramo, mientras que la distancia de un vector al origen viene dada por la norma euclidiana, o norma ℓ_2 : $\|\mathbf{x}\| = \sqrt{(\sum_i x_i^2)}$.

La norma al cuadrado $\|\mathbf{x}\|^2$ es de especial interés, ya que corresponde a la suma de los elementos al cuadrado. La operación vectorial fundamental que nos interesa es el producto interno, o producto escalar, que se obtiene multiplicando los dos vectores elemento a elemento y sumando los valores resultantes.

Definición, producto interno: el producto interno entre dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ viene dado por la expresión $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_i x_i y_i$.

La notación con corchetes angulares es común en física, y la usamos a veces por claridad. Es importante notar que el producto escalar entre dos vectores es un escalar. Por ejemplo, si $\mathbf{x} = [0.1, 0, -0.3]$ e $\mathbf{y} = [-4.0, 0.05, 0.1]$, entonces $\langle \mathbf{x}, \mathbf{y} \rangle = -0.4 + 0 - 0.03 = -0.43$.

$$\begin{array}{r}
 \mathbf{x} \quad \begin{array}{|c|c|c|} \hline 0.1 & 0 & -0.3 \\ \hline \end{array} \\
 \mathbf{y} \quad \begin{array}{|c|c|c|} \hline -4 & 0.05 & 0.1 \\ \hline \end{array} \\
 \hline
 -0.4 \quad 0 \quad -0.03 = -0.43
 \end{array}$$

$$\mathbf{x}^T \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \alpha$$

SÍMILITUD COS.

-1



Una interpretación geométrica simple del producto escalar la da su relación con el ángulo α entre los dos vectores: $\mathbf{x}^T \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos(\alpha)$.

Por tanto, para dos vectores normalizados, cuya norma es igual a 1, el producto escalar equivale al coseno de su ángulo, y en ese caso llamamos al producto escalar similitud coseno. La similitud coseno $\cos(\alpha)$ oscila entre 1, para dos vectores que apuntan en la misma dirección, y -1 , para dos vectores que apuntan en direcciones opuestas, con el caso especial de $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ que da lugar a vectores ortogonales que apuntan en direcciones perpendiculares. Mirándolo desde otro ángulo, para dos vectores normalizados, de norma unitaria, una vez que fijamos el primer argumento $\mathbf{x}$ obtenemos lo que sigue.

Preguntas

1. ¿Por qué sumar un tensor de forma (4, 1) a uno de forma (4) da un resultado (4, 4)?
 - a. Porque la suma de vectores siempre produce una matriz
 - b. Porque la difusión alinea las formas desde la derecha y repite cada tensor para igualarlas, convirtiendo la suma elemento a elemento en una tabla de 4 por 4
 - c. Porque el marco traspone la columna automáticamente
 - d. Porque uno de los tensores contiene enteros
2. ¿Qué es el producto interno de dos vectores?
 - a. Un escalar: la suma de los productos elemento a elemento
 - b. Un vector de los productos elemento a elemento

- c. Una matriz construida a partir de los dos vectores
 - d. El ángulo entre los dos vectores
3. Para dos vectores normalizados, ¿qué significa un producto interno de 0?
- a. Apuntan en la misma dirección
 - b. Apuntan en direcciones opuestas
 - c. Son ortogonales, apuntan en direcciones perpendiculares
 - d. Uno de ellos es el vector cero
4. ¿Qué entiende el libro por el rango n de un tensor?
- a. El número de sus elementos
 - b. El número de sus dimensiones: 0 para escalares, 1 para vectores, 2 para matrices
 - c. Su valor más grande
 - d. El número de tensores que lo componen

Respuestas

- 1. b)** La difusión alinea las formas desde la derecha y repite los tensores para igualarlas, así que una columna más un tensor unidimensional se convierte en una matriz completa: probablemente no era lo que se pretendía.
- 2. a)** El producto interno multiplica los vectores elemento a elemento y suma los resultados, así que es un solo número.
- 3. c)** Para vectores unitarios el producto escalar es el coseno de su ángulo, y un coseno de 0 significa vectores perpendiculares, ortogonales.
- 4. b)** Un tensor es una matriz de n dimensiones; $n = 0$ da escalares, $n = 1$ vectores, $n = 2$ matrices.

Ejemplos resueltos

Un producto escalar a mano Calcule el producto interno de $x = [0.1, 0, -0.3]$ e $y = [-4.0, 0.05, 0.1]$.

Multiplique los vectores elemento a elemento: $0.1 \times (-4.0) = -0.4$, $0 \times 0.05 = 0$ y $-0.3 \times 0.1 = -0.03$.

Sume los tres productos, ya que el producto interno es la suma de los productos elemento a elemento.

$-0.4 + 0 - 0.03 = -0.43$; el resultado es un escalar, como todo producto escalar.

Respuesta: -0.43

Ejercicios

1. Halle la norma euclidiana de $x = [3, 4]$ y su norma al cuadrado.

Pista: Eleve al cuadrado los elementos, súmelos y luego saque la raíz cuadrada.

Respuesta: La norma al cuadrado es $3^2 + 4^2 = 9 + 16 = 25$, y la norma es $\sqrt{25} = 5$.

2. Se dan los vectores unitarios $x = [1, 0]$ e $y = [0, 1]$. ¿Cuál es su similitud coseno y qué dice sobre sus direcciones?

Pista: Para vectores unitarios el producto escalar es igual al coseno del ángulo.

Respuesta: Su producto escalar es $1 \times 0 + 0 \times 1 = 0$, así que la similitud coseno es 0: los vectores son ortogonales, apuntan en direcciones perpendiculares.

3. Suma un tensor de forma (3, 1) a un tensor de forma (3). ¿Qué forma da la difusión, y cómo obtendría en su lugar una suma elemento a elemento de forma (3)?

Pista: La difusión alinea las formas desde la derecha y repite para igualarlas.

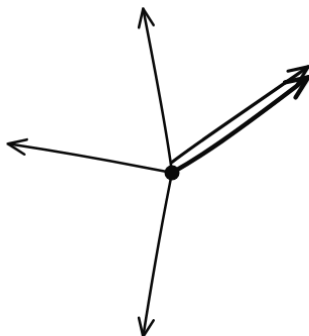
Respuesta: El resultado tiene forma $(3, 3)$, porque la columna se repite en 3 columnas y el tensor unidimensional en 3 filas. Para obtener una simple suma elemento a elemento, dé a ambos tensores la misma forma, por ejemplo cambiando primero la forma del tensor $(3, 1)$ a (3) .

Operaciones matriciales comunes

$\mathbf{y}^* = \operatorname{argmax} \langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}$, donde argmax denota la operación de hallar el valor de $\mathbf{y}$ que corresponde al mayor valor de salida posible. De la ecuación E.2.3 vemos que, para maximizar el producto escalar, el segundo vector debe ser igual al primero. Esto es importante, porque en los capítulos siguientes $\mathbf{x}$ representará una entrada, mientras que $\mathbf{w}$ representará parámetros adaptables, de modo que el producto escalar se maximiza siempre que $\mathbf{x}$ «resuena» con $\mathbf{w}$, lo que se llama coincidencia de plantilla.

$$\langle \mathbf{x}, \mathbf{1} \rangle = \sum_i x_i$$

RESUENA
 $\mathbf{W} = \mathbf{X}$: MÁX


$$\|\mathbf{x} - \mathbf{y}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle$$
$$\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$$

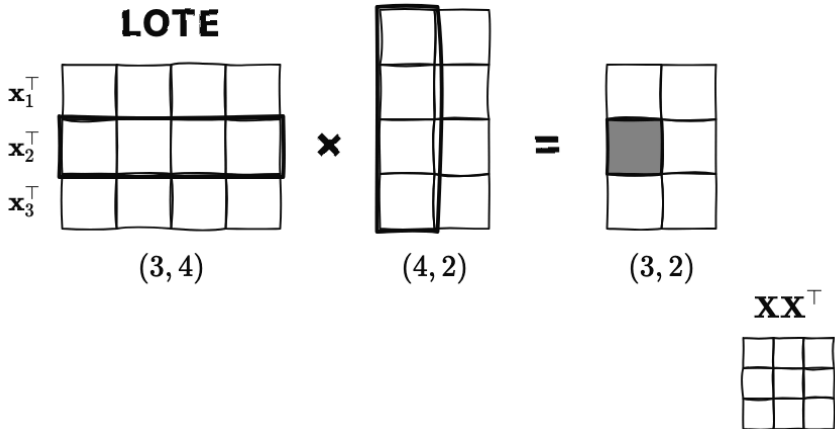
La mejor coincidencia es él mismo

Cerramos con dos observaciones adicionales que serán útiles.

Primero, podemos escribir la suma de los elementos de un vector como su producto escalar con un vector $\mathbf{1}$ compuesto enteramente de unos, $\mathbf{1} = [1, 1, \dots, 1]^T$: $\langle \mathbf{x}, \mathbf{1} \rangle = \sum_{i=1}^d x_i$. Segundo, la distancia entre dos vectores también puede escribirse en términos de sus productos escalares: $\|\mathbf{x} - \mathbf{y}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle$. El caso $\mathbf{y} = \mathbf{0}$ nos da $\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle$. Ambas ecuaciones pueden ser útiles al escribir ecuaciones o en el código.

Operaciones matriciales comunes. En el caso bidimensional tenemos matrices: $\mathbf{X} \sim (n, d)$, una cuadrícula de elementos desde X_{11} en la esquina superior izquierda hasta X_{nd} en la inferior derecha. En este caso podemos hablar de una matriz con n filas y d columnas. De especial importancia para lo que sigue, una matriz puede entenderse como una pila de n vectores $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, donde la pila se organiza por filas: las filas de $\mathbf{X}$ son desde $\mathbf{x}_1^T$ hasta $\mathbf{x}_n^T$.

PROD. ESCALAR



Filas por columnas

Decimos que $\mathbf{X}$ representa un lote de vectores de datos. Como veremos, es habitual definir los modelos, tanto matemáticamente como en código, para que trabajen sobre datos en lotes de este tipo. Una operación fundamental para las matrices es la multiplicación.

Definición, multiplicación de matrices: para dos matrices cualesquiera $\mathbf{X} \sim (a, b)$ e $\mathbf{Y} \sim (b, c)$ de forma compatible, la multiplicación de matrices $\mathbf{Z} = \mathbf{XY}$, con $\mathbf{Z} \sim (a, c)$, se define elemento a elemento como $Z_{ij} = \langle \mathbf{X}_i, \mathbf{Y}_j^T \rangle$. Es decir, el elemento (i, j) del producto es el producto escalar entre la fila i -ésima de $\mathbf{X}$ y la columna j -ésima de $\mathbf{Y}$.

Como caso especial, si el segundo término es un vector tenemos un producto matriz-vector: $\mathbf{z} = \mathbf{W}\mathbf{x}$.

Si interpretamos $\mathbf{X}$ como un lote de vectores, la multiplicación de matrices $\mathbf{X}\mathbf{W}^T$ es una forma vectorizada simple de calcular n productos escalares como en la ecuación E.2.5, uno por cada fila de $\mathbf{X}$, con una sola operación de álgebra lineal. Como otro ejemplo, la multiplicación de una matriz por su traspuesta, $\mathbf{X}\mathbf{X}^T \sim (n, n)$, es una forma vectorizada de calcular a la vez todos los productos escalares posibles de pares de filas de $\mathbf{X}$.

Cerramos mencionando algunas operaciones adicionales sobre matrices que serán importantes.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \odot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$$

$\exp(\mathbf{X})$

e^1	e^2
e^3	e^4

$$\text{mat-exp}(\mathbf{X}) = \sum_k \mathbf{X}^k / k!$$

SOLO CUADRADAS
MÓDULO LINALG

Elemento a elemento

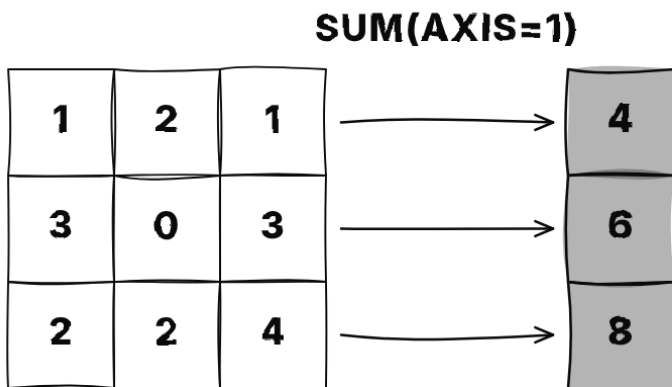
Definición, multiplicación de Hadamard: para dos matrices de la misma forma, la multiplicación de Hadamard se define elemento a elemento como $[\mathbf{X} \odot \mathbf{Y}]_{i,j} = X_{i,j} Y_{i,j}$.

Aunque la multiplicación de Hadamard no tiene todas las propiedades algebraicas interesantes de la multiplicación de matrices estándar, se usa habitualmente en modelos diferenciables para realizar operaciones de enmascaramiento, por ejemplo poner algunos elementos a cero, o de escalado. Las interacciones multiplicativas también se han vuelto populares en algunas familias recientes de modelos, como veremos a continuación.

A veces escribimos expresiones como $\exp(\mathbf{X})$, que deben interpretarse como aplicaciones elemento a elemento de la operación: $[\exp(\mathbf{X})]_{i,j} = \exp(X_{i,j})$.

En comparación, la «verdadera» exponenciación matricial se define para una matriz cuadrada como $\text{mat-exp}(\mathbf{X}) = \sum_{k=0}^{\infty} \mathbf{X}^k / k!$.

Es importante notar que la ecuación E.2.6 puede definirse para tensores de cualquier forma, mientras que E.2.7 solo es válida para matrices cuadradas. Por eso todos los marcos, como PyTorch, tienen módulos especializados que reúnen todas las operaciones específicas de matrices, como inversas y determinantes. El recuadro C.2.2 da un ejemplo, contrastando la exponencial elemento a elemento de una matriz con la exponencial matricial de los libros de álgebra lineal: en PyTorch la primera es la función `exp` ordinaria, la segunda vive en el subpaquete de álgebra lineal como `matrix_exp`. Las operaciones especializadas de álgebra lineal se encapsulan generalmente en su propio subpaquete.



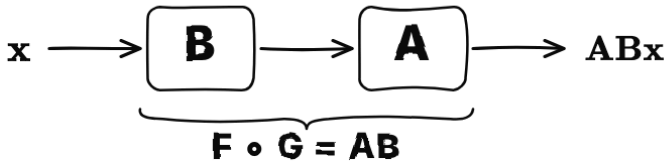
Reducir a lo largo de un eje

Por último, podemos escribir operaciones de reducción, como la suma o la media, a lo largo de ejes sin especificar índices inferior y superior, en cuyo caso suponemos que la suma recorre el eje completo: $\sum_i \mathbf{X}_i = \sum_{i=1}^n \mathbf{X}_i$. En PyTorch y otros marcos, las operaciones de reducción corresponden a métodos con un argumento de eje, por ejemplo una suma a lo largo del eje 1.

Un recuadro del libro pregunta: sobre la definición de la multiplicación de matrices, ¿por qué se define como en la ecuación E.2.4 y no como la multiplicación de Hadamard? Considere un vector $\mathbf{x}$ y alguna función genérica f definida sobre él. Se dice que la función es lineal si $f(\alpha \mathbf{x}_1 + \beta \mathbf{x}_2) = \alpha f(\mathbf{x}_1) + \beta f(\mathbf{x}_2)$. Cualquier función así puede representarse como una matriz $\mathbf{A}$; esto se ve extendiendo los dos

vectores en una representación en una base. Entonces el producto matriz-vector $\mathbf{Ax}$ corresponde a la aplicación de la función, $f(\mathbf{x}) = \mathbf{Ax}$, y la multiplicación de matrices $\mathbf{AB}$ corresponde a la composición de funciones $f \circ g$, donde $(f \circ g)(\mathbf{x}) = f(g(\mathbf{x}))$ y $g(\mathbf{x}) = \mathbf{Bx}$.

$$f(\alpha \mathbf{x}_1 + \beta \mathbf{x}_2) = \alpha f(\mathbf{x}_1) + \beta f(\mathbf{x}_2)$$



Las matrices son funciones lineales

Preguntas

1. Para vectores unitarios, ¿qué $\mathbf{y}$ maximiza el producto escalar $\langle \mathbf{x}, \mathbf{y} \rangle$?
 - a. $\mathbf{y} = -\mathbf{x}$
 - b. $\mathbf{y} = \mathbf{x}$

- c. Cualquier y ortogonal a $\mathbf{x}$
 - d. El vector cero
2. ¿Cuál es el elemento (i, j) del producto matricial $\mathbf{XY}$?
- a. $X_{i,j}$ por $Y_{i,j}$
 - b. La suma de la fila i de $\mathbf{X}$
 - c. El producto escalar de la fila i -ésima de $\mathbf{X}$ con la columna j -ésima de $\mathbf{Y}$
 - d. El producto escalar de la columna i -ésima de $\mathbf{X}$ con la fila j -ésima de $\mathbf{Y}$
3. ¿Qué calcula el producto $\mathbf{XX}^T$ para un lote $\mathbf{X}$ de n vectores fila?
- a. Todos los productos escalares entre pares de filas, en una matriz de n por n
 - b. La suma de todas las filas
 - c. Los cuadrados elemento a elemento de $\mathbf{X}$
 - d. La norma de cada columna
4. ¿Por qué la multiplicación de matrices se define fila por columna y no elemento a elemento?
- a. Porque es más rápida en las GPU
 - b. Porque una función lineal es una matriz, así que el producto matricial corresponde a componer dos funciones lineales
 - c. Porque el producto de Hadamard solo funciona para vectores
 - d. Porque evita errores de difusión

Respuestas

1. b) Para maximizar el producto escalar, el segundo vector debe ser igual al primero; por eso una entrada «resuena» con los parámetros que coinciden

con ella, la coincidencia de plantilla.

2. c) La multiplicación de matrices se define elemento a elemento como el producto escalar entre la fila i de la primera matriz y la columna j de la segunda.

3. a) Multiplicar una matriz por su traspuesta es una forma vectorizada de calcular a la vez todos los productos escalares de pares de sus filas.

4. b) Cualquier función lineal puede representarse como una matriz; el producto matriz-vector es aplicarla, y el producto de matrices es la composición de funciones.

Ejemplos resueltos

Producto matricial frente a producto de Hadamard Sean $X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ e $Y = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$. Calcule el producto matricial XY y el producto de Hadamard $X \odot Y$.

Para el producto matricial, cada elemento es una fila de X multiplicada escalarmente por una columna de Y .

Fila 1 con columna 1: $1 \times 5 + 2 \times 7 = 19$; fila 1 con columna 2: $1 \times 6 + 2 \times 8 = 22$.

Fila 2 con columna 1: $3 \times 5 + 4 \times 7 = 43$; fila 2 con columna 2: $3 \times 6 + 4 \times 8 = 50$.

Para el producto de Hadamard, multiplique los elementos correspondientes: $1 \times 5 = 5$, $2 \times 6 = 12$, $3 \times 7 = 21$, $4 \times 8 = 32$.

Respuesta: $XY = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$, mientras que $X \odot Y = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$.

Ejercicios

1. Use el vector de unos para hallar la suma de los elementos de $x = [2, -1, 4]$.

Pista: La suma de los elementos es igual a $\langle x, 1 \rangle$.

Respuesta: $\langle x, 1 \rangle = 2 \times 1 + (-1) \times 1 + 4 \times 1 = 5$.

2. Para $x = [1, 2]$ e $y = [3, 1]$, calcule la distancia al cuadrado $\|x - y\|^2$ usando solo productos escalares, y luego compruébelo directamente.

Pista: Use $\|x - y\|^2 = \langle x, x \rangle + \langle y, y \rangle - 2\langle x, y \rangle$.

Respuesta: $\langle x, x \rangle = 5$, $\langle y, y \rangle = 10$, $\langle x, y \rangle = 3 + 2 = 5$, así que $5 + 10 - 10 = 5$. Directamente, $x - y = [-2, 1]$ y $(-2)^2 + 1^2 = 5$.

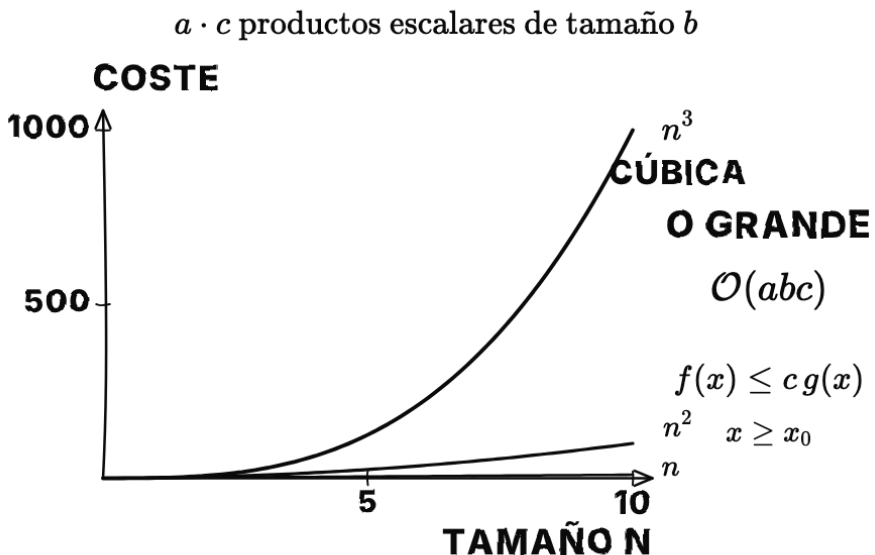
3. Una matriz X tiene forma $(10, 3)$ y W tiene forma $(4, 3)$. ¿Cuál es la forma de XW^T , y qué contiene cada elemento?

Pista: Trasponga W primero y luego compruebe las dimensiones interiores.

Respuesta: W^T tiene forma $(3, 4)$, así que XW^T tiene forma $(10, 4)$. Cada elemento (i, j) es el producto escalar de la fila i de X con la fila j de W .

Complejidad computacional

Usaré la multiplicación de matrices para introducir el tema de la complejidad de una operación. Mirando la ecuación E.2.4, vemos que calcular la matriz $Z \sim (a, c)$ a partir de los argumentos de entrada $X \sim (a, b)$ e $Y \sim (b, c)$ requiere ac productos internos de dimensión b si aplicamos directamente la definición, que es lo que llamamos complejidad temporal, mientras que el requisito de memoria para una implementación secuencial es simplemente el tamaño de la matriz de salida, que llamamos en cambio complejidad espacial.



Cómo crece el coste con el tamaño

Para abstraerse de los detalles concretos del hardware, la informática se centra en la llamada notación $\mathcal{O}$, del alemán Ordnung, que significa orden de aproximación. Se dice que una función $f(x)$ es $\mathcal{O}(g(x))$, donde suponemos que tanto las entradas como las salidas son no negativas, si podemos encontrar una constante c y un valor x_0 tales que $f(x) \leq c \cdot g(x)$ para cualquier $x \geq x_0$. Esto significa que, en cuanto x crece lo suficiente, podemos ignorar en nuestro análisis todos los factores fuera de $g(x)$. Esto se llama análisis asintótico. Por tanto, podemos decir que una implementación ingenua de la multiplicación de matrices es $\mathcal{O}(abc)$, creciendo linealmente respecto a los tres parámetros de entrada. Para dos matrices cuadradas de tamaño (n, n) , decimos que la multiplicación de matrices es cúbica en la dimensión de entrada.

Razonar en términos de complejidad asintótica es importante, y elegante, pero elegir un algoritmo solo por su complejidad $\mathcal{O}$ no se traduce necesariamente en ganancias prácticas de rendimiento, que dependen de muchos detalles, como qué hardware se usa, qué paralelismo se admite, y así sucesivamente. Cuando llama a una primitiva concreta en un marco de álgebra lineal, como la multiplicación de matrices en PyTorch, la implementación de bajo nivel que se ejecuta, el kernel, depende del hardware en tiempo de ejecución, mediante un proceso conocido como despacho; por eso el mismo código puede ejecutarse mediante un kernel de GPU, un kernel de CPU, un kernel de TPU, y así sucesivamente. Esto se vuelve aún más complejo con compiladores como XLA, que pueden optimizar el código fusionando operaciones pensando en un hardware objetivo concreto. Como ejemplo, se sabe que el mejor algoritmo asintótico para multiplicar dos matrices cuadradas de tamaño (n, n) escala como $\mathcal{O}(n^c)$ para una constante $c < 2.4$, lo que es mucho mejor

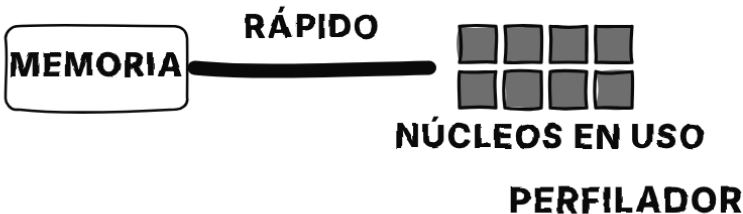
que el requisito cúbico $\mathcal{O}(n^3)$ de una implementación ingenua. Sin embargo, estos algoritmos son mucho más difíciles de paralelizar eficientemente en hardware altamente paralelo como las GPU, lo que los hace poco comunes en la práctica.

$\mathcal{O}(n^{2.37})$ va mal en las GPU

MEMORY-BOUND



COMPUTE-BOUND



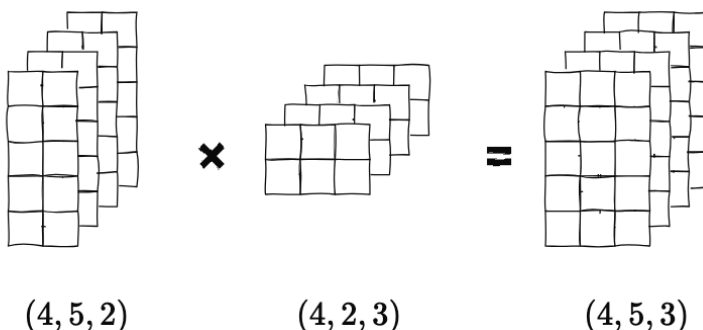
Limitado por memoria o por cómputo

Observe que, desde el punto de vista de la complejidad asintótica, tener acceso a un entorno paralelo con k procesadores no tiene impacto, ya que solo puede ofrecer, en el mejor de los casos, una aceleración constante de $1/k$ respecto a una implementación no paralela. Además, la complejidad asintótica no tiene en cuenta el tiempo que lleva mover los datos de un lugar a otro, que puede convertirse en el principal cuello de botella en muchas situaciones. En estos casos, decimos que la implementación está limitada por la

memoria, en oposición a limitada por el cómputo. En la práctica, esto solo puede comprobarse ejecutando un perfilador sobre el código. Veremos que analizar la complejidad de un algoritmo está lejos de ser trivial debido a la interacción entre la complejidad asintótica y la complejidad observada.

Operaciones tensoriales de orden superior. Los vectores y las matrices son interesantes porque nos permiten definir un gran número de operaciones que no están definidas o son complejas en dimensiones superiores, por ejemplo exponenciales matriciales, multiplicación de matrices y determinantes. Al pasar a dimensiones superiores, la mayoría de las operaciones que nos interesan son o bien variantes por lotes de operaciones matriciales, o bien combinaciones concretas de operaciones matriciales y operaciones de reducción.

$$[\text{BMM}(\mathbf{X}, \mathbf{Y})]_i = \mathbf{X}_i \mathbf{Y}_i$$



LOTE PRIMERO

Multiplicación de matrices por lotes

Como ejemplo de lo primero, considere dos tensores $\mathbf{X} \sim (n, a, b)$ e $\mathbf{Y} \sim (n, b, c)$. La multiplicación de matrices por lotes, BMM, se define como $[\text{BMM}(\mathbf{X}, \mathbf{Y})]_i = \mathbf{X}_i \mathbf{Y}_i \sim (n, a, c)$.

Las operaciones en la mayoría de los marcos actúan de forma transparente sobre versiones por lotes de sus argumentos, que se supone, como en este caso, que son las dimensiones principales, las primeras. Por ejemplo, la multiplicación de matrices por lotes en PyTorch es igual que la multiplicación de matrices estándar, como muestra el recuadro C.2.3: multiplicar un tensor de forma 4, 5, 2 por uno de forma 4, 2, 3 da un resultado de forma 4, 5, 3. La mayoría de las operaciones pueden trabajar con entradas posiblemente por lotes.

Como ejemplo de operación de reducción, considere dos tensores $X, Y \sim (a, b, c)$. Una versión generalizada del producto escalar, GDT, puede escribirse como $\text{GDT}(X, Y) = \sum_{i,j,k} [X \otimes Y]_{i,j,k}$, que es simplemente un producto escalar sobre las versiones «aplanadas» de sus entradas. Este breve repaso cubre la mayoría de las operaciones tensoriales que usaremos en el resto del libro, con material adicional introducido cuando sea necesario.

Preguntas

1. ¿Por qué se dice que una multiplicación ingenua de matrices de $X \sim (a, b)$ por $Y \sim (b, c)$ es $\mathcal{O}(abc)$?
 - a. Porque necesita $a \cdot c$ productos internos, cada uno de dimensión b
 - b. Porque la salida tiene $a \cdot b \cdot c$ elementos
 - c. Porque necesita $a + b + c$ operaciones
 - d. Porque usa tres procesadores
2. ¿Por qué los algoritmos más rápidos que $\mathcal{O}(n^3)$ para la multiplicación de matrices rara vez se usan en la práctica?
 - a. Dan resultados erróneos
 - b. Necesitan más memoria de la que tiene cualquier computador
 - c. Son mucho más difíciles de paralelizar eficientemente en las GPU
 - d. Solo funcionan para matrices pequeñas

3. ¿Qué significa que una implementación esté limitada por la memoria?
 - a. Su tiempo de ejecución está dominado por mover datos y no por calcular
 - b. No puede ejecutarse sin una GPU
 - c. Su complejidad O grande es cúbica
 - d. Usa menos memoria que una limitada por el cómputo
4. ¿Por qué tener k procesadores no cambia la complejidad O grande?
 - a. Porque los procesadores no pueden compartir datos
 - b. Porque da como mucho una aceleración constante de $1/k$, y O grande ignora los factores constantes
 - c. Porque O grande solo se aplica a las GPU
 - d. Porque k procesadores siempre son más lentos

Respuestas

1. **a)** Aplicar la definición directamente requiere ac productos internos, cada uno sobre b elementos, así que el trabajo crece como abc .
2. **c)** Existen algoritmos asintóticamente mejores, con exponente por debajo de 2.4, pero se paralelizan mal en hardware altamente paralelo como las GPU.
3. **a)** El análisis asintótico ignora el movimiento de datos, que puede convertirse en el principal cuello de botella; entonces la implementación está limitada por la memoria, y solo un perfilador puede decirlo.
4. **b)** El paralelismo proporciona a lo sumo un factor constante de $1/k$, y la notación asintótica descarta las constantes.

Ejemplos resueltos

Contar el trabajo de un producto matricial ¿Cuántos productos internos, y cuántas multiplicaciones escalares, necesita una implementación ingenua para multiplicar $X \sim (100, 200)$ por $Y \sim (200, 50)$?

La salida Z tiene forma $(a, c) = (100, 50)$, así que hay $100 \times 50 = 5000$ elementos que calcular.

Cada elemento es un producto interno de dimensión $b = 200$, que necesita 200 multiplicaciones.

Multiplicaciones totales: $5000 \times 200 = 1\,000\,000$, que es exactamente $a \cdot b \cdot c$, en concordancia con $\mathcal{O}(abc)$.

Respuesta: 5000 productos internos y 1 000 000 de multiplicaciones.

Ejercicios

1. Dos matrices cuadradas de tamaño (n, n) se multiplican de forma ingenua. Si n se duplica, ¿en qué factor crece el trabajo?

Pista: El algoritmo ingenuo es cúbico en n .

Respuesta: En $2^3 = 8$, ya que el trabajo crece como n^3 .

2. En PyTorch, un tensor de forma $(8, 3, 4)$ se multiplica matricialmente por un tensor de forma $(8, 4, 6)$. ¿Cuál es la forma del resultado y por qué?

Pista: La dimensión principal se trata como un lote.

Respuesta: $(8, 3, 6)$: la primera dimensión es un lote de 8, y cada par de rebanadas, $(3, 4)$ por $(4, 6)$, da una matriz $(3, 6)$.

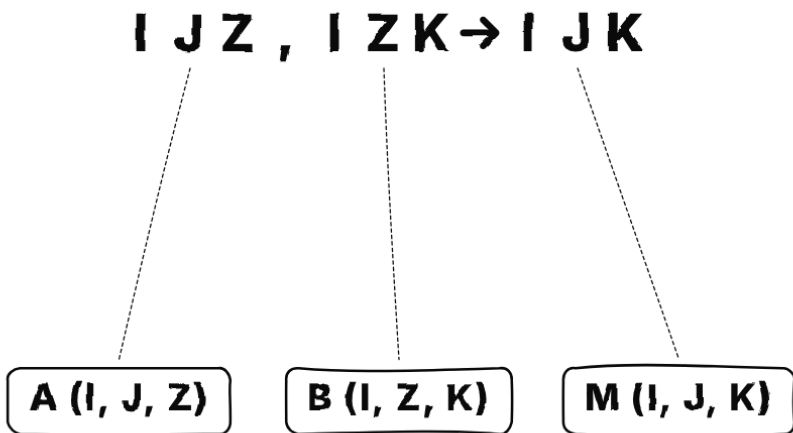
3. Explique con sus propias palabras por qué $\text{GDT}(X, Y)$ para dos tensores de forma (a, b, c) es simplemente un producto escalar.

Pista: Imagine cada tensor extendido como un único vector largo.

Respuesta: GDT multiplica los tensores elemento a elemento y lo suma todo; si ambos tensores se aplanan en vectores de longitud abc en el mismo orden, eso es exactamente la suma de los productos elemento a elemento, el producto escalar ordinario.

La notación de Einstein y las derivadas

Esta es una sección opcional que trata einsum, un conjunto de convenciones que permite al usuario especificar prácticamente cualquier operación tensorial, incluidas reducciones, sumas y multiplicaciones, con una sintaxis simple basada en cadenas de texto. El recuadro C.2.4 muestra dos ejemplos de einsum en PyTorch: una multiplicación de matrices por lotes, escrita con la cadena «i jota zeta, i zeta ka, a i jota ka», y un producto escalar generalizado, escrito «i jota ka, i jota ka, a nada».



PYTORCH = JAX

Leer una cadena einsum

Para introducir la notación, consideremos de nuevo los dos ejemplos mostrados antes en las ecuaciones E.2.9 y E.2.10, escribiendo explícitamente todos los ejes: $M_{i j k} = \sum_z A_{i j z} B_{z k}$, y $M = \sum_i \sum_j \sum_k X_{i j k} Y_{i j k}$.

En línea con la notación de Einstein, en una versión simplificada que ignora la distinción entre índices superiores e inferiores, podemos simplificar las dos ecuaciones eliminando las sumas, bajo la convención de que cualquier índice que aparece a la derecha pero no a la izquierda se suma. Así, $M_{i j k} = A_{i j z} B_{z k}$ representa la suma sobre z , y $M = X_{i j k} Y_{i j k}$ representa la suma sobre i, j y k .

Entonces podemos condensar las dos definiciones aislando los índices en una sola cadena, donde los operandos están ahora a la izquierda: « i jota zeta, i zeta k a, a i jota k a» para la multiplicación de matrices por lotes, e « i jota k a, i jota k a, a nada» para el producto escalar generalizado.

Hay una correspondencia directa uno a uno entre las definiciones de las ecuaciones E.2.13 y E.2.14 y sus definiciones simplificadas en cadenas. Esto está implementado en la mayoría de los marcos en la operación `einsum`, como muestra el recuadro C.2.4.

La ventaja de esta notación es que no necesitamos recordar la API de un marco para implementar una operación dada, y traducir de un marco a otro es transparente porque la sintaxis de `einsum` es equivalente. Por ejemplo, PyTorch tiene varios métodos de multiplicación de matrices, incluidos `matmul` y `bmm`, con distintas reglas de difusión y restricciones de forma, y `einsum` ofrece una sintaxis uniforme para todos ellos. Además, la definición en `einsum` de nuestra multiplicación de matrices por lotes es idéntica, por

ejemplo, a la definición en JAX, como muestra el recuadro C.2.5, con exactamente la misma cadena.

Trabajar con ejes traspuestos también es simple. Por ejemplo, para $A \sim (n, a, b)$ y $B \sim (n, c, b)$, una multiplicación por lotes de $[A]_i$ por $[B^T]_i$ se obtiene intercambiando los ejes correspondientes en la definición de einsum, a «i jota zeta, i ka zeta, a i jota ka».

$$\begin{array}{c} \partial \\ ij\,z, \, i\,zk \rightarrow ijk \\ \updownarrow \\ ij\,z, \, i\,kz \rightarrow ijk \end{array}$$

GRADIENTES Y JACOBIANOS

Trasponer intercambiando letras

Por estas razones, einsum y sus generalizaciones, como el popular paquete einops, han ganado recientemente una amplia popularidad.

Sección 2.2, Gradientes y jacobianos. Como implica el nombre diferenciable, los gradientes desempeñan un papel central en el libro, ofreciendo una forma de optimizar nuestros modelos mediante

mecanismos semiautomáticos derivados del descenso de gradiente. Recordamos aquí algunas definiciones y conceptos básicos sobre la diferenciación de funciones multivaluadas. Nos centramos en propiedades que serán esenciales más adelante, en parte a expensas de la precisión matemática.

Derivadas de funciones escalares. Partiendo de una función simple $y = f(x)$ con una entrada escalar y una salida escalar, su derivada se define como sigue.

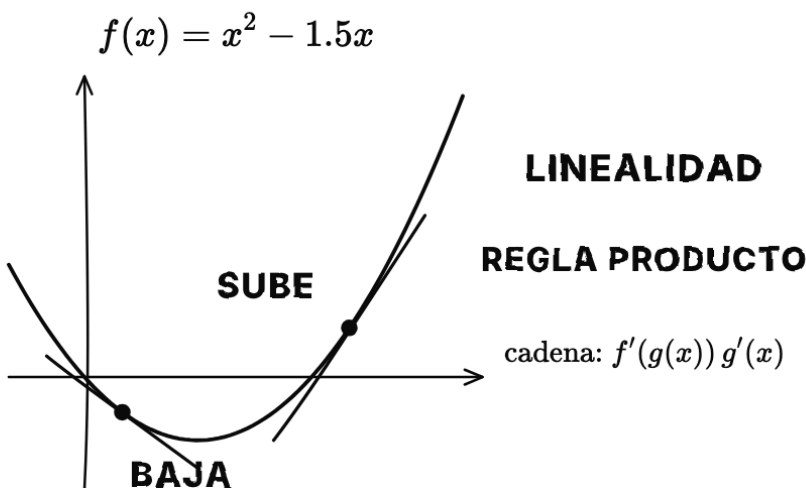
Definición, derivada: la derivada de $f(x)$ se define como $f'(x) = \lim_{h \rightarrow 0} (f(x + h) - f(x)) / h$.

Usamos diversas notaciones para denotar derivadas. ∂ denotará genéricamente derivadas y gradientes de cualquier dimensión, vectores y matrices; ∂_x o $\partial/\partial x$ resalta el argumento de entrada respecto al cual derivamos, cuando hace falta; mientras que $f'(x)$ es específica de funciones escalares, y a veces se llama notación de Lagrange.

No nos ocupamos aquí de la existencia de la derivada de la función, que no está garantizada en todas partes ni siquiera para una función continua, y que damos por supuesta. Solo tocaremos este punto al discutir derivadas de funciones no suaves, como el valor absoluto $f(x) = \text{abs}(x)$ en 0, más adelante en el capítulo 6.

Las derivadas de funciones simples pueden obtenerse por aplicación directa de la definición; por ejemplo, las derivadas de un polinomio, un logaritmo o un seno deberían resultar familiares: $\partial x^p = p x^{p-1}$, $\partial \log(x) = 1/x$, y $\partial \sin(x) = \cos(x)$.

Geométricamente, la derivada puede entenderse como la pendiente de la tangente que pasa por un punto, o equivalentemente como la mejor aproximación de primer orden de la propia función en ese punto, como muestra la figura F.2.1. Este es un punto de vista fundamental, porque la pendiente de la recta nos dice cómo evoluciona la función en un entorno cercano: para una pendiente positiva, la función crece a la derecha y decrece a la izquierda, de nuevo en un intervalo suficientemente pequeño, mientras que para una pendiente negativa ocurre lo contrario. Como veremos, esta intuición se extiende a las funciones vectoriales. La figura dibuja la función $f(x) = x^2 - 1.5x$, junto con sus derivadas en dos puntos distintos.



Pendientes de $x^2 - 1.5x$

Recordamos algunas propiedades importantes de las derivadas que se extienden al caso multidimensional. Linealidad: la derivada es lineal, así que la derivada de una suma es la suma de las derivadas, $\partial[f(x) + g(x)] = f'(x) + g'(x)$. La regla del producto: $\partial[f(x)g(x)] = f'(x)g(x) + f(x)g'(x)$. La regla de la cadena: la derivada de una composición de funciones se obtiene multiplicando las derivadas correspondientes, $\partial[f(g(x))] = f'(g(x)) g'(x)$.

Preguntas

1. En la notación de Einstein tal como la usa el libro, ¿qué índices se suman?
 - a. Todo índice que aparece dos veces a la izquierda
 - b. Cualquier índice que aparece a la derecha del signo igual pero no a la izquierda
 - c. Solo el índice z
 - d. Ninguno; las sumas siempre deben escribirse
2. ¿Qué calcula la cadena $\text{einsum } \langle i j k, i j k, a \text{ nada} \rangle$?
 - a. Una multiplicación de matrices por lotes
 - b. Una traspuesta
 - c. Un producto escalar generalizado: todos los índices se suman y queda un solo número
 - d. Un producto elemento a elemento conservado como tensor
3. ¿Qué le dice una derivada positiva en un punto sobre la función en sus cercanías?

- a. La función crece a la derecha y decrece a la izquierda, en un intervalo suficientemente pequeño
 - b. La función es positiva ahí
 - c. La función tiene un máximo ahí
 - d. La función es constante en las cercanías
4. ¿Qué dice la regla de la cadena sobre la derivada de $f(g(x))$?
- a. Es igual a $f'(x) + g'(x)$
 - b. Es igual a $f'(g(x))$ por $g'(x)$
 - c. Es igual a $f'(x)$ por $g'(x)$
 - d. Es igual a $f(g'(x))$

Respuestas

- 1. b)** La convención elimina los signos de suma: un índice presente a la derecha pero ausente a la izquierda se suma.
- 2. c)** No queda nada en el lado de la salida, así que los tres índices se suman: el producto escalar generalizado.
- 3. a)** La derivada es la pendiente de la tangente; una pendiente positiva significa que la función sube a la derecha y baja a la izquierda, localmente.
- 4. b)** La derivada de una composición es el producto de las derivadas, con la exterior evaluada en $g(x)$.

Ejemplos resueltos

Pendientes de la función de ejemplo del libro Para $f(x) = x^2 - 1.5x$, la función dibujada en la figura F.2.1, halle la derivada y su valor en $x = 0$ y en $x = 1$. Diga si f crece o decrece cerca de cada punto.

Derive término a término, usando la linealidad y la regla de las potencias: la derivada de x^2 es $2x$ y la derivada de $1.5x$ es 1.5 .

Así, $f'(x) = 2x - 1.5$.

En $x = 0$: $f'(0) = -1.5$, una pendiente negativa, así que f decrece cerca de 0.

En $x = 1$: $f'(1) = 2 - 1.5 = 0.5$, una pendiente positiva, así que f crece cerca de 1.

Respuesta: $f'(x) = 2x - 1.5$; $f'(0) = -1.5$ (decrece), $f'(1) = 0.5$ (crece).

Ejercicios

1. Use la regla del producto para derivar $f(x) = x \cdot \sin(x)$.

Pista: Tome el primer factor como x y el segundo como $\sin(x)$.

Respuesta: $f'(x) = 1 \cdot \sin(x) + x \cdot \cos(x) = \sin(x) + x \cos(x)$.

2. Use la regla de la cadena para derivar $h(x) = \log(x^2)$ para $x > 0$, y simplifique.

Pista: La función exterior es $\log$, la interior x^2 .

Respuesta: $h'(x) = (1/x^2) \cdot 2x = 2/x$.

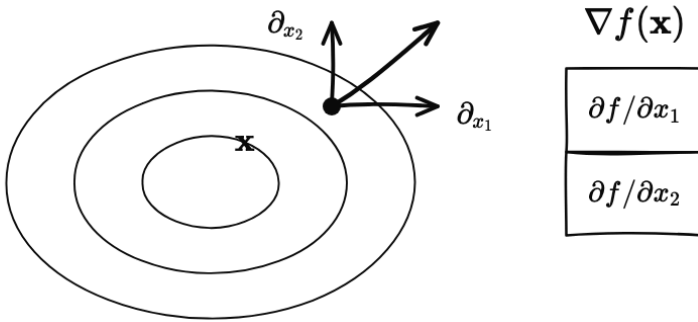
3. Escriba una cadena einsum para el producto matriz-vector $z = Wx$, donde W tiene índices i, j y x tiene índice j .

Pista: ¿Qué índice desaparece de la salida?

Respuesta: « i, j, j, a »: j aparece en el lado de las entradas pero no en la salida, así que se suma, dando $z_i = \sum_j W_{ij} x_j$.

Gradientes, derivadas direccionales y jacobianos

Considere ahora una función $y = f(\mathbf{x})$ que toma como entrada un vector $\mathbf{x} \sim (d)$. Hablar aquí de perturbaciones infinitesimales no tiene sentido a menos que especifiquemos la dirección de la perturbación: mientras que en el caso escalar solo teníamos «izquierda» y «derecha», en este caso tenemos infinitas direcciones posibles en el espacio euclidiano. En el caso más simple, podemos considerar movernos a lo largo del eje i -ésimo, manteniendo fijos todos los demás valores: $\partial_{x_i} f(\mathbf{x}) = \partial y / \partial x_i = \lim_{h \rightarrow 0} (f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})) / h$. Aquí $\mathbf{e}_i \sim (d)$ es el i -ésimo vector de la base, la fila i -ésima de la matriz identidad: su elemento j -ésimo es 1 si $i = j$, y 0 en otro caso.



$$\mathbf{e}_i = [0, 1, 0]$$

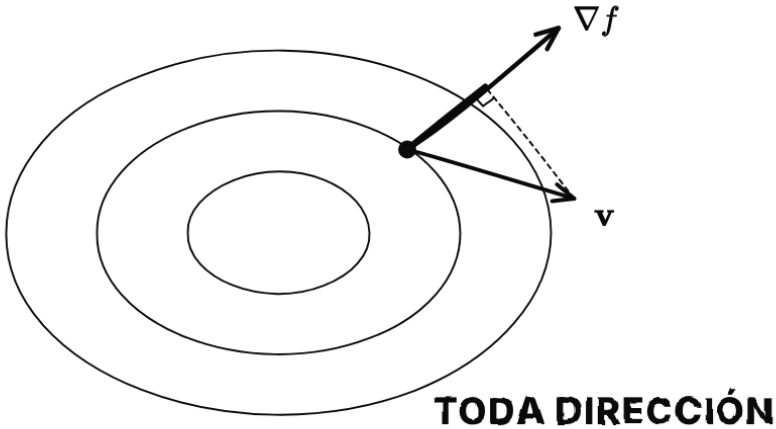
Las derivadas parciales forman un gradiente

La ecuación E.2.17 se llama derivada parcial. Apilar todas las derivadas parciales nos da un vector de d dimensiones llamado el gradiente de la función.

Definición, gradiente: el gradiente de una función $y = f(\mathbf{x})$ es el vector columna $\nabla f(\mathbf{x}) = \partial f(\mathbf{x})$, cuyas entradas son las derivadas parciales desde $\partial_{x_1} f(\mathbf{x})$ hasta $\partial_{x_d} f(\mathbf{x})$.

Como los gradientes son fundamentales, usamos la notación especial $\nabla f(\mathbf{x})$ para distinguirlos. ¿Y qué pasa con los desplazamientos en una dirección general $\mathbf{v}$? En este caso obtenemos la derivada direccional: $D_{\mathbf{v}} f(\mathbf{x}) = \lim_{h \rightarrow 0} (f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})) / h$.

$$D_{\mathbf{v}}f(\mathbf{x}) = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})}{h} = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$$



Cualquier dirección desde el gradiente

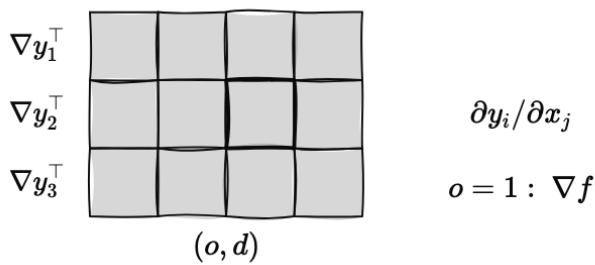
El movimiento en el espacio puede descomponerse en desplazamientos individuales a lo largo de cada eje; por tanto es fácil demostrar que la derivada direccional viene dada por el producto escalar del gradiente con el vector de desplazamiento $\mathbf{v}$: $D_{\mathbf{v}}f(\mathbf{x}) = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle = \sum_i \partial_{x_i} f(\mathbf{x}) v_i$, donde cada v_i es el desplazamiento en el eje i -ésimo.

Por tanto, saber calcular el gradiente de una función basta para calcular todas las derivadas direccionales posibles.

Jacobianos. Consideremos ahora el caso genérico de una función $\mathbf{y} = f(\mathbf{x})$ con una entrada vectorial $\mathbf{x} \sim (d)$ como antes, y esta vez una salida vectorial $\mathbf{y} \sim (o)$. Como veremos, este es el caso más general

que necesitamos considerar. Como tenemos más de una salida, calculamos un gradiente para cada salida, y su pila nos da una matriz (o, d) llamada el jacobiano de f.

$$\mathbf{x} \sim (d) \rightarrow f \rightarrow \mathbf{y} \sim (o)$$



$$\partial[f \circ g] = \partial f \cdot \partial g$$



El jacobiano: gradientes apilados

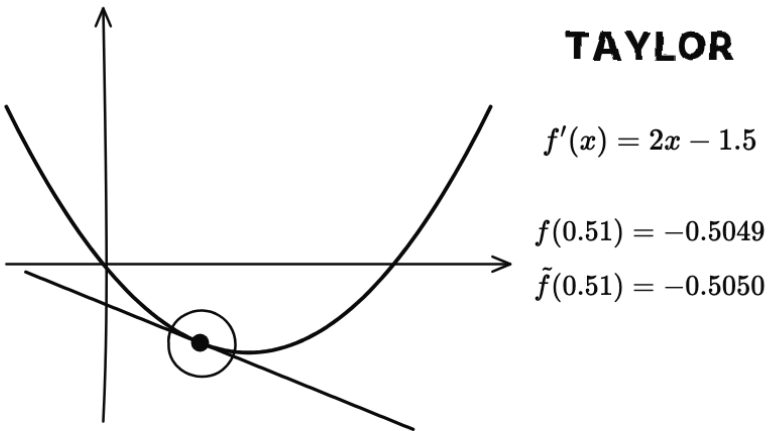
Definición, jacobiano: la matriz jacobiana de una función $\mathbf{y} = f(\mathbf{x})$, con $\mathbf{x} \sim (d)$ e $\mathbf{y} \sim (o)$, es la matriz $\partial f(\mathbf{x}) \sim (o, d)$ cuyo elemento en la fila i y la columna j es $\partial y_i / \partial x_j$, desde $\partial y_1 / \partial x_1$ en la esquina superior izquierda hasta $\partial y_o / \partial x_d$ en la inferior derecha.

Recuperamos el gradiente para $o = 1$, y la derivada estándar para $d = o = 1$. Los jacobianos heredan todas las propiedades de las derivadas: es importante que el jacobiano de una composición de funciones es ahora una multiplicación de matrices de los jacobianos individuales

correspondientes, $\partial[f(g(\mathbf{x}))] = [\partial f(\bullet)] \partial g(\mathbf{x})$, donde el primer jacobiano se evalúa en $g(\mathbf{x}) \sim (h)$.

Como en el caso escalar, los gradientes y los jacobianos pueden entenderse como funciones lineales tangentes en un punto concreto. En particular, el gradiente es la mejor «aproximación de primer orden» en el sentido siguiente. Para un punto $\mathbf{x}_0$, la mejor aproximación lineal en un entorno infinitesimal de $f(\mathbf{x}_0)$ viene dada por $\tilde{f}(\mathbf{x}) = f(\mathbf{x}_0) + \langle \partial f(\mathbf{x}_0), \mathbf{x} - \mathbf{x}_0 \rangle$, donde el gradiente hace el papel de la pendiente de la recta, y $\mathbf{x} - \mathbf{x}_0$ es el desplazamiento desde $\mathbf{x}_0$.

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}_0) + \langle \partial f(\mathbf{x}_0), \mathbf{x} - \mathbf{x}_0 \rangle$$



La tangente como aproximación

Esto se llama el teorema de Taylor. El recuadro C.2.6 y la figura F.2.2 lo visualizan en el caso escalar $f(x) = x^2 - 1.5x$. El recuadro escribe la

función, su derivada $2x - 1.5$, calculada a mano por ahora, y la linealización en $x = 0.5$. Como comprobación numérica, en $x = 0.51$ la función da -0.5049 y la linealización da -0.5050 .

Preguntas

1. ¿Cómo se obtiene la derivada direccional de f a lo largo de cualquier dirección $\mathbf{v}$ una vez que se conoce el gradiente?
 - a. Tomando el producto escalar del gradiente con $\mathbf{v}$
 - b. Multiplicando el gradiente solo por la longitud de $\mathbf{v}$
 - c. Tomando la mayor entrada del gradiente
 - d. Hay que calcular un nuevo límite para cada dirección
2. ¿Cuál es la forma del jacobiano de una función de vectores de tamaño d a vectores de tamaño o ?
 - a. (d, o)
 - b. (o, d)
 - c. (d)
 - d. (o)
3. ¿Cómo es la regla de la cadena para los jacobianos?
 - a. El jacobiano de una composición es la suma de los jacobianos
 - b. Es el producto de Hadamard de los jacobianos
 - c. Es el producto matricial de los jacobianos individuales, con el exterior evaluado en $\mathbf{g}(\mathbf{x})$
 - d. Solo vale para funciones escalares

4. En la comprobación del libro, ¿por qué $f(0.51) = -0.5049$ y su linealización -0.5050 están tan cerca?
- a. Porque f es lineal
 - b. Porque la linealización es la mejor aproximación de primer orden cerca del punto 0.5 , y 0.51 está muy cerca de él
 - c. Porque la derivada es cero en 0.5
 - d. Porque el redondeo oculta la diferencia

Respuestas

1. **a)** El movimiento se descompone a lo largo de los ejes, así que la derivada direccional es $\langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$; el gradiente basta para toda dirección.
2. **b)** Hay una fila gradiente por cada una de las o salidas, cada una con d entradas, así que el jacobiano es o por d .
3. **c)** Los jacobianos heredan las propiedades de las derivadas; para una composición se multiplican como matrices.
4. **b)** El teorema de Taylor dice que la recta tangente es la mejor aproximación lineal en un entorno pequeño; un paso de 0.01 se queda en ese entorno.

Ejemplos resueltos

La linealización del libro en 0.5 Para $f(x) = x^2 - 1.5x$, construya la aproximación de primer orden en $x = 0.5$ y compárela con f en $x = 0.51$.

Evalúe la función en el punto: $f(0.5) = 0.25 - 0.75 = -0.5$.

Evalúe la derivada $2x - 1.5$ en el punto: $2 \times 0.5 - 1.5 = -0.5$, la pendiente de la tangente.

La linealización es $\tilde{f}(x) = -0.5 + (-0.5)(x - 0.5)$; en 0.51 el desplazamiento es 0.01 , así que $\tilde{f}(0.51) = -0.5 - 0.005 = -0.505$.

El valor verdadero es $f(0.51) = 0.2601 - 0.765 = -0.5049$, que difiere solo en 0.0001.

Respuesta: $\tilde{f}(0.51) = -0.5050$ frente a $f(0.51) = -0.5049$.

Ejercicios

1. Para $f(x_1, x_2) = x_1^2 + 3x_1x_2$, halle el gradiente en $(1, 2)$ y la derivada direccional a lo largo de $v = (1, -1)$.

Pista: Derive respecto a cada variable manteniendo fija la otra, y luego tome un producto escalar con v .

Respuesta: $\partial f / \partial x_1 = 2x_1 + 3x_2 = 2 + 6 = 8$ y $\partial f / \partial x_2 = 3x_1 = 3$, así que el gradiente es $(8, 3)$. A lo largo de v : $8 \times 1 + 3 \times (-1) = 5$.

2. Halle el jacobiano de $y = (x_1x_2, x_1 + x_2)$ en el punto $(2, 3)$, y dé su forma.

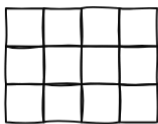
Pista: Una fila por salida, una columna por entrada.

Respuesta: Fila 1: $(\partial(x_1x_2)/\partial x_1, \partial(x_1x_2)/\partial x_2) = (x_2, x_1) = (3, 2)$. Fila 2: $(1, 1)$. El jacobiano es $[[3, 2], [1, 1]]$, de forma $(2, 2)$.

Las dimensiones de los jacobianos y el descenso de gradiente

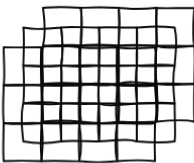
Cerramos con una nota pedante sobre dimensionalidad que será útil en lo que sigue. Considere la función $y = \mathbf{W}\mathbf{x}$. Vista como función de $\mathbf{x}$, la derivada es, como antes, una matriz (o, d) , y puede demostrarse que $\partial_{\mathbf{x}}[\mathbf{W}\mathbf{x}] = \mathbf{W}$.

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$



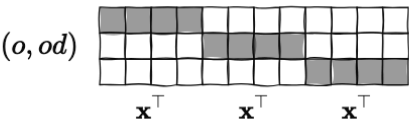
$$\partial_{\mathbf{x}}[\mathbf{W}\mathbf{x}] = \mathbf{W}$$

(o, d)



$$\partial_{\mathbf{W}}[\mathbf{W}\mathbf{x}] \sim (o, o, d)$$

VALORES REPET.



VECTOR-JACOB.

Jacobianos de $y = \mathbf{W}\mathbf{x}$

La figura F.2.2 muestra la función $f(\mathbf{x}) = \mathbf{x}^2 - 1.5\mathbf{x}$ y su aproximación de primer orden en 0.5.

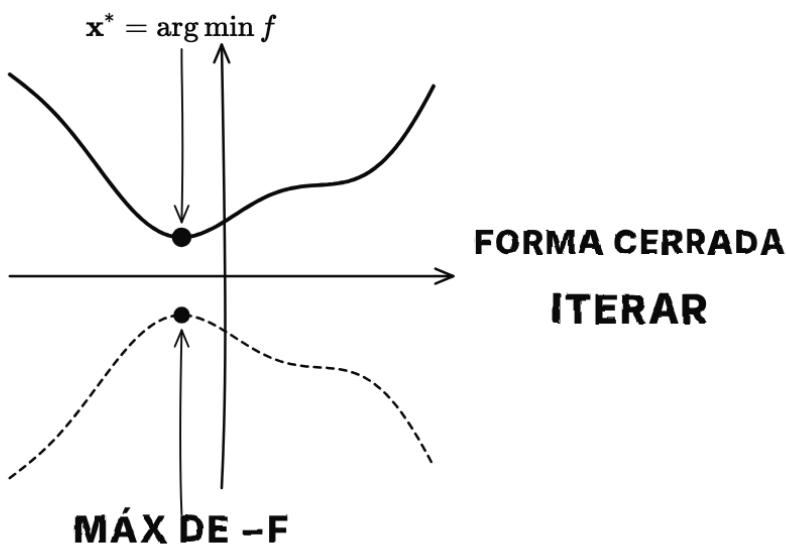
Vista como función de $\mathbf{W}$, en cambio, la entrada es ella misma una matriz (o, d), y el «jacobiano» en este caso tiene forma (o, o, d), como muestra el recuadro que sigue. Sin embargo, siempre podemos imaginar una función idéntica, isomorfa, que toma como entrada la versión vectorizada de $\mathbf{W}$, $\text{vect}(\mathbf{W}) \sim (\text{od})$, en cuyo caso el jacobiano será una matriz de forma (o, od).

El recuadro desarrolla el jacobiano. Para calcular el jacobiano $\partial_{\mathbf{W}} \mathbf{W} \mathbf{x}$, podemos reescribir la expresión elemento a elemento como $y_i = \sum_j W_{ij} x_j$, de donde hallamos inmediatamente que $\partial y_i / \partial W_{ij} = x_j$. Observe que para materializar el jacobiano explícitamente, es decir, para guardarlo en memoria, necesitaríamos muchos valores repetidos. Como veremos en el capítulo 6, esto puede evitarse porque en la práctica solo nos importa la aplicación del jacobiano a otro tensor.

Este ejemplo rápido aclara lo que queremos decir con nuestra afirmación de que trabajar con entradas y salidas vectoriales basta desde el punto de vista de la notación. Sin embargo, será importante tener este punto en mente en el capítulo 6, cuando usemos jacobianos matriciales por simplicidad de notación, en particular para evitar la proliferación de índices, pero los tamaños de estos jacobianos pueden «escondarse» dentro de las formas reales de las entradas y las salidas, sobre todo los tamaños de lote. Veremos en el capítulo 6 que el cálculo explícito de los jacobianos puede evitarse en la práctica considerando los llamados productos vector-jacobiano. Esto también puede formalizarse viendo los jacobianos como aplicaciones lineales abstractas.

Sección 2.3, Descenso de gradiente. Para entender la utilidad de tener acceso a los gradientes, considere el problema de minimizar una

función genérica $f(x)$, con $x \sim (d)$: $x^* = \operatorname{argmin}_x f(x)$. Aquí, de modo similar a argmax , $\operatorname{argmin} f(x)$ denota la operación de hallar el valor de x que corresponde al menor valor posible de $f(x)$. Suponemos que la función tiene una sola salida, lo que es optimización de un solo objetivo, y que el dominio sobre el que optimizamos x no tiene restricciones.



Buscando el punto más bajo

En el resto del libro x codificará los parámetros de nuestro modelo, y f describirá el rendimiento del propio modelo sobre nuestros datos, un planteamiento llamado aprendizaje supervisado que presentamos en el próximo capítulo. Podemos considerar minimizar en lugar de maximizar sin pérdida de generalidad, ya que minimizar $f(x)$ equivale a maximizar $-f(x)$ y viceversa; para visualizarlo, piense en una

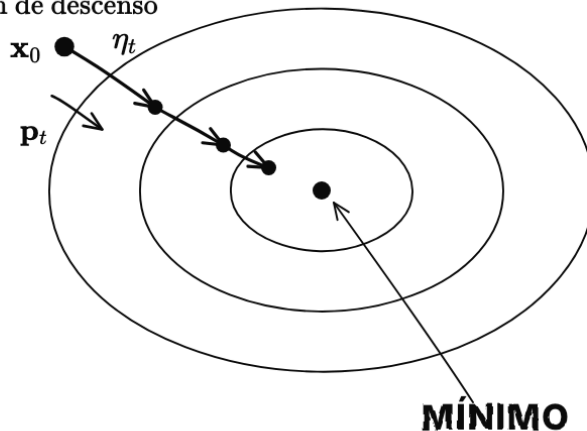
función de una dimensión y voltéela respecto al eje x, imaginando qué pasa con sus puntos bajos.

En casos muy raros, podemos expresar la solución en forma cerrada; veremos un ejemplo en el contexto de la optimización por mínimos cuadrados en la sección 4.1.2. En general, sin embargo, nos vemos obligados a recurrir a procedimientos iterativos. Suponga que partimos de una conjetura aleatoria $\mathbf{x}_0$ y que, en cada iteración, damos un paso, que descomponemos en su magnitud η_t , la longitud del paso, y la dirección $\mathbf{p}_t$: $\mathbf{x}_t = \mathbf{x}_{t-1} + \eta_t \mathbf{p}_t$, siendo la conjetura en la iteración t la conjetura anterior más el desplazamiento en la iteración t.

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \eta_t \mathbf{p}_t$$

η_t : tasa de aprendizaje

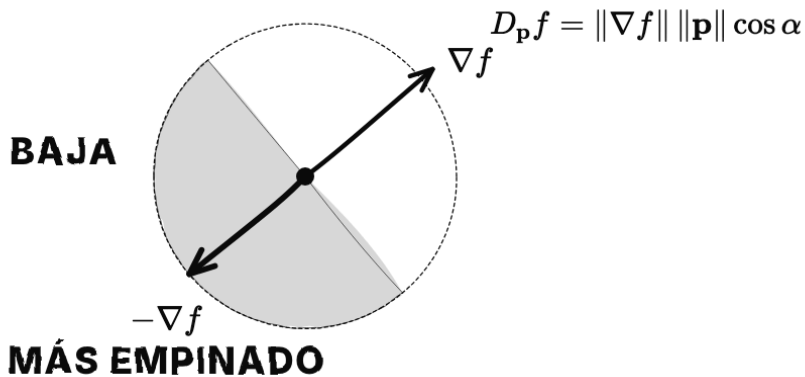
$\mathbf{p}_t$: dirección de descenso



Paso a paso cuesta abajo

Llamamos a η_t el tamaño del paso o, en la terminología del aprendizaje automático, la tasa de aprendizaje, por razones que quedarán claras en el próximo capítulo. Una dirección $\mathbf{p}_t$ para la que existe un η_t tal que $f(\mathbf{x}_t) \leq f(\mathbf{x}_{t-1})$ se llama dirección de descenso. Si podemos seleccionar una dirección de descenso en cada iteración, y si somos cuidadosos en la elección del tamaño del paso, el algoritmo iterativo de la ecuación E.2.26 convergerá a un mínimo, en un sentido que describiremos en breve.

Para funciones diferenciables, podemos cuantificar con precisión todas las direcciones de descenso usando la derivada direccional de la ecuación E.2.20, ya que pueden definirse como las direcciones que inducen un cambio negativo respecto a nuestra conjetura anterior $\mathbf{x}_{t-1}$: si $\mathbf{p}_t$ es una dirección de descenso, entonces $D_{\mathbf{p}_t} f(\mathbf{x}_{t-1}) \leq 0$.



Qué direcciones bajan

Usando lo que aprendimos en la sección 2.2 y la definición del producto escalar en términos de la similitud coseno de la ecuación E.2.2, obtenemos $D_{\mathbf{p}_t} f(\mathbf{x}_{t-1}) = \langle \nabla f(\mathbf{x}_{t-1}), \mathbf{p}_t \rangle = \|\nabla f(\mathbf{x}_{t-1})\| \|\mathbf{p}_t\| \cos(\alpha)$, donde α es el ángulo entre $\mathbf{p}_t$ y $\nabla f(\mathbf{x}_{t-1})$.

Considerando la expresión de la derecha, el primer término es una constante respecto a $\mathbf{p}_t$. Como hemos supuesto que $\mathbf{p}_t$ solo codifica la dirección del movimiento, también podemos restringirlo sin problema a $\|\mathbf{p}_t\| = 1$, convirtiendo el segundo término en otra constante. Por tanto, por las propiedades del coseno, deducimos que cualquier $\mathbf{p}_t$ cuyo ángulo con $\nabla f(\mathbf{x}_{t-1})$ esté entre $\pi/2$ y $3\pi/2$ es una dirección de descenso. Entre ellas, la dirección $\mathbf{p}_t = -\nabla f(\mathbf{x}_{t-1})$, con

un ángulo de π , tiene la menor derivada direccional posible, y nos referimos a ella como la dirección de descenso más pronunciado.

Juntar esta idea con el procedimiento iterativo de la ecuación E.2.26 nos da un algoritmo para minimizar cualquier función diferenciable, que llamamos descenso de gradiente más pronunciado.

Definición, descenso de gradiente más pronunciado: dada una función diferenciable $f(\mathbf{x})$, un punto de partida $\mathbf{x}_0$ y una secuencia de tamaños de paso η_t , el descenso de gradiente procede como $\mathbf{x}_t = \mathbf{x}_{t-1} - \eta_t \nabla f(\mathbf{x}_{t-1})$.

No nos ocuparemos del problema de encontrar un tamaño de paso apropiado, que simplemente supondremos «lo bastante pequeño» para que la iteración del descenso de gradiente proporcione una reducción de f . En la próxima sección nos centramos en qué puntos se obtienen al ejecutar el descenso de gradiente desde una inicialización genérica. Observe que el descenso de gradiente es tan eficiente como el procedimiento que usamos para calcular el gradiente: presentamos un algoritmo general para ello en el capítulo 6.

Preguntas

1. Para $\mathbf{y} = \mathbf{W}\mathbf{x}$ vista como función de $\mathbf{W} \sim (o, d)$, ¿qué forma tiene el jacobiano?
 - a. (o, d)
 - b. (o, o, d) , o (o, od) si $\mathbf{W}$ se aplana
 - c. (d, o)
 - d. (o)

2. ¿Qué dirección $\mathbf{p}_t$ tiene la menor derivada direccional, el descenso más pronunciado?
 - a. El propio gradiente
 - b. Cualquier dirección en ángulo recto con el gradiente
 - c. Menos el gradiente, en un ángulo de π con él
 - d. Cualquier vector unitario
3. ¿Por qué el libro puede estudiar solo la minimización, sin pérdida de generalidad?
 - a. Porque la maximización es imposible para funciones diferenciables
 - b. Porque minimizar f es lo mismo que maximizar $-f$
 - c. Porque todas las funciones tienen un único mínimo
 - d. Porque los gradientes solo apuntan cuesta abajo
4. ¿Por qué el libro evita materializar en memoria jacobianos como $\partial_{\mathbf{W}} \mathbf{W}\mathbf{x}$?
 - a. Contendrían muchos valores repetidos, y en la práctica solo se necesita su aplicación a otro tensor
 - b. No pueden calcularse
 - c. Siempre son cero
 - d. Solo están definidos para escalares

Respuestas

1. **b)** La entrada es ella misma una matriz o por d, así que el jacobiano tiene forma (o, o, d) ; aplanar $\mathbf{W}$ en un vector de od entradas lo convierte en una matriz (o, od) .

2. c) Con las normas fijas, la derivada direccional es proporcional a $\cos(\alpha)$, que es mínimo, -1 , en $\alpha = \pi$: el gradiente negativo.

3. b) Voltear la función respecto al eje x convierte sus puntos altos en puntos bajos, así que un problema puede convertirse en el otro.

4. a) Las entradas del jacobiano son solo los x_j repetidos, y el capítulo 6 muestra que solo se necesitan productos con él, los productos vector-jacobiano.

Ejemplos resueltos

Dos pasos de descenso de gradiente Minimice $f(x) = x^2$ por descenso de gradiente desde $x_0 = 3$ con un tamaño de paso constante $\eta = 0.1$. Calcule dos pasos y el valor de f en cada punto.

El gradiente de x^2 es $2x$, y la actualización es $x_{t+1} = x_t - \eta \cdot 2x_t$.

Paso 1: el gradiente en 3 es 6, así que $x_1 = 3 - 0.1 \times 6 = 2.4$.

Paso 2: el gradiente en 2.4 es 4.8, así que $x_2 = 2.4 - 0.1 \times 4.8 = 1.92$.

Los valores de la función bajan de $f(3) = 9$ a $f(2.4) = 5.76$ y a $f(1.92) = 3.6864$, así que cada paso es un descenso.

Respuesta: $x_1 = 2.4$, $x_2 = 1.92$; f va $9 \rightarrow 5.76 \rightarrow 3.6864$.

Ejercicios

1. En cierto punto el gradiente de f es $(2, -1)$. ¿Es $p = (1, 1)$ una dirección de descenso? ¿Y $p = (-1, 0)$?

Pista: Calcule la derivada direccional $\langle \nabla f, p \rangle$ y compruebe su signo.

Respuesta: Para $(1, 1)$: $2 \times 1 + (-1) \times 1 = 1 > 0$, así que no es una dirección de descenso. Para $(-1, 0)$: $2 \times (-1) + 0 = -2 < 0$, así que es una dirección de descenso.

2. Para $y = Wx$ con W de forma $(3, 4)$, dé la forma del jacobiano respecto a x , respecto a W y respecto a la W aplanada.

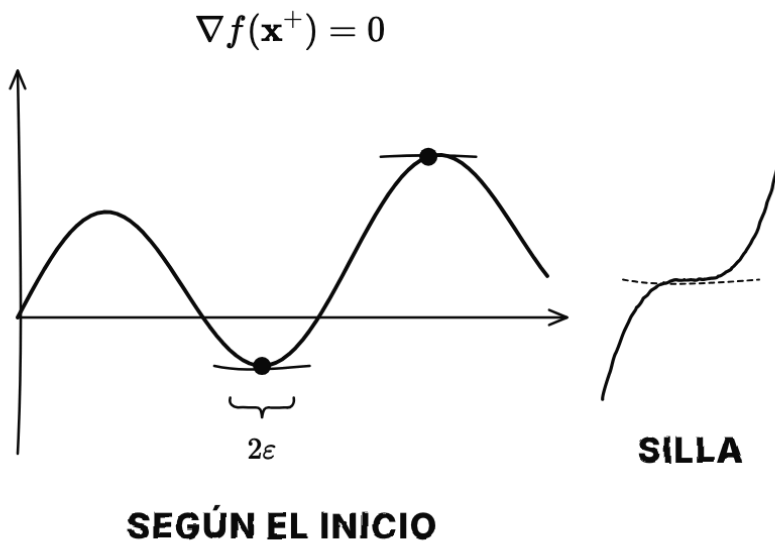
Pista: Respecto a x es la propia W .

Respuesta: Respecto a x : $(3, 4)$, igual a W . Respecto a W : $(3, 3, 4)$.

Respecto a $\text{vect}(W)$, un vector de 12 entradas: $(3, 12)$.

Convergencia del descenso de gradiente

Al hablar de la convergencia del descenso de gradiente, tenemos que aclarar qué entendemos por «un minimizador» de una función. Si no le importa la convergencia y confía en el descenso de gradiente, pase sin dudar a la siguiente sección.



Mínimos, máximos y sillas

Definición, mínimo: un mínimo local de $f(\mathbf{x})$ es un punto $\mathbf{x}^+$ tal que, para algún $\varepsilon > 0$, $f(\mathbf{x}^+) \leq f(\mathbf{x})$ para todo $\mathbf{x}$ con $\|\mathbf{x} - \mathbf{x}^+\| < \varepsilon$, es decir, para todo punto de una bola de tamaño ε centrada en $\mathbf{x}^+$.

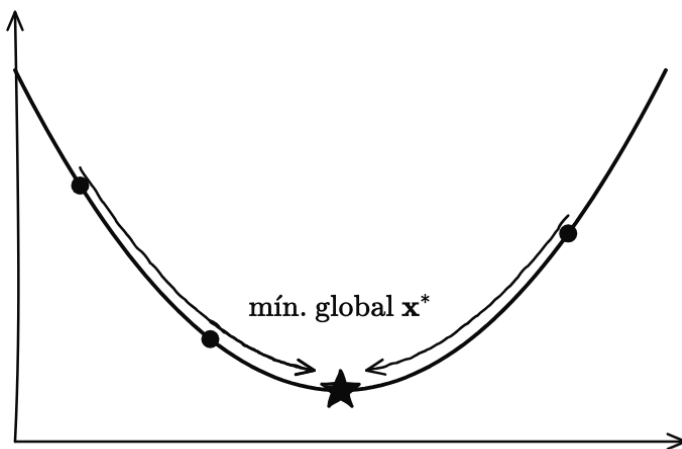
En palabras, el valor $f(\mathbf{x}^+)$ es un mínimo si consideramos un entorno suficientemente pequeño de $\mathbf{x}^+$. Intuitivamente, en tal punto la pendiente de la tangente será 0, y el gradiente en todo el resto del entorno de $\mathbf{x}^+$ apuntará hacia arriba. Podemos formalizar la primera idea con el concepto de puntos estacionarios.

Definición, puntos estacionarios: un punto $\mathbf{x}^+$ se llama punto estacionario de $f(\mathbf{x})$ si $\nabla f(\mathbf{x}^+) = \mathbf{0}$.

Los puntos estacionarios no se limitan a los mínimos: pueden ser máximos, que son los mínimos de $-f(\mathbf{x})$, o puntos de silla, que son puntos de inflexión donde la curvatura de la función está cambiando, como en el ejemplo de la figura F.2.3. En general, sin ninguna restricción sobre f , solo puede demostrarse que el descenso de gradiente converge a un punto estacionario genérico, según su inicialización.

La figura es un ejemplo simple de punto de silla; intente visualizar la recta tangente en ese punto para ver que, en efecto, es estacionario.

¿Podemos hacerlo mejor? Imagine una parábola: en este caso, la función no tiene ningún punto de silla y solo tiene un único mínimo. Este mínimo también es especial, en el sentido de que en ese punto la función alcanza su menor valor posible en todo el dominio: decimos que es un mínimo global.

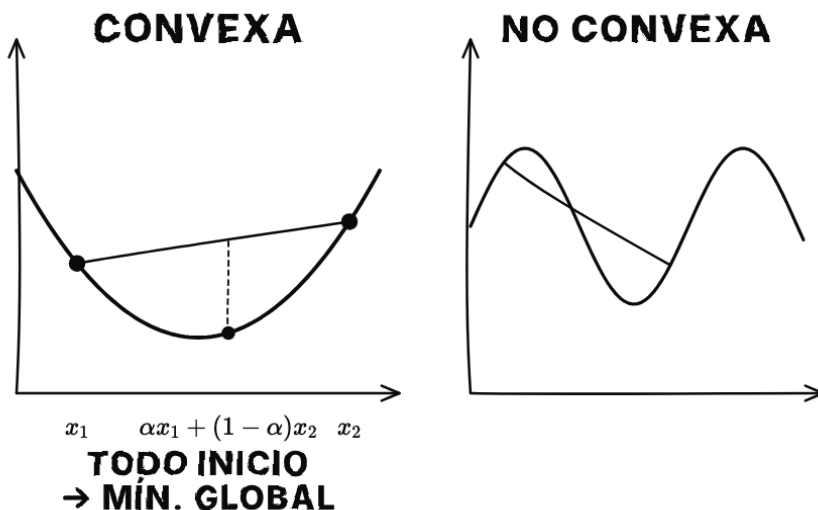


Una parábola tiene un mínimo global

Definición, mínimo global: un mínimo global de $f(\mathbf{x})$ es un punto $\mathbf{x}^*$ tal que $f(\mathbf{x}^*) \leq f(\mathbf{x})$ para cualquier entrada posible $\mathbf{x}$.

Intuitivamente, el descenso de gradiente convergerá a este mínimo global si se ejecuta sobre una parábola, desde cualquier inicialización posible, porque todos los gradientes apuntarán hacia él. Podemos generalizar esta idea con el concepto de convexidad de una función. Hay muchas definiciones posibles de convexidad; elegimos la de abajo por simplicidad de exposición.

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2)$$



Convexa: bajo toda cuerda

Definición, función convexa: una función $f(x)$ es convexa si, para dos puntos cualesquiera x_1 y x_2 y cualquier $\alpha \in [0, 1]$, tenemos $f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2)$.

El lado izquierdo de la ecuación E.2.28 es el valor de f en cualquier punto dentro del intervalo que va de x_1 a x_2 , mientras que el lado derecho es el valor correspondiente sobre el segmento de recta que une $f(x_1)$ y $f(x_2)$. Si la función está siempre por debajo de la recta que une dos puntos cualesquiera, es convexa; como ejemplo, una parábola que apunta hacia arriba es convexa.

La convexidad califica la simplicidad de optimizar la función, en el sentido siguiente. Para una función genérica no convexa, el descenso

de gradiente converge a un punto estacionario, y nada más puede decirse a menos que miremos las derivadas de orden superior, las derivadas de las derivadas. Para una función convexa, el descenso de gradiente convergerá a un mínimo global, independientemente de la inicialización. Si la desigualdad de la ecuación E.2.28 se cumple de forma estricta, lo que es la convexidad estricta, el minimizador global será además único.

Esta es una propiedad dura: para encontrar un mínimo global en un problema no convexo con descenso de gradiente, la única solución es ejecutar el optimizador infinitas veces desde cada inicialización posible, lo que lo convierte en una tarea NP-difícil.

Esta discusión tiene un fuerte significado histórico. Como veremos en el capítulo 5, cualquier modelo no trivial es no convexo, lo que significa que su problema de optimización puede tener varios puntos estacionarios. Esto contrasta con algoritmos alternativos de aprendizaje supervisado, como las máquinas de vectores de soporte, que mantienen la no linealidad permitiendo a la vez una optimización convexa. Curiosamente, los modelos diferenciables complejos parecen funcionar bien incluso frente a esta restricción, en el sentido de que su optimización, cuando se inicia desde una inicialización razonable, converge a puntos con buen rendimiento empírico.

Preguntas

1. Sin ninguna restricción sobre f , ¿a qué puede demostrarse que llega el descenso de gradiente?
 - a. Siempre a un mínimo global

- b. A un punto estacionario genérico, que depende de la inicialización
 - c. Siempre a un máximo local
 - d. A nada en absoluto
2. ¿Qué garantiza la convexidad para el descenso de gradiente?
- a. Convergencia a un mínimo global, sea cual sea la inicialización
 - b. Convergencia en exactamente un paso
 - c. Convergencia a un punto de silla
 - d. Que el gradiente nunca es cero
3. ¿Por qué la no convexidad de las redes neuronales es históricamente significativa?
- a. Hizo imposible entrenar redes neuronales
 - b. Contrasta con métodos como las máquinas de vectores de soporte, que mantienen la no linealidad con optimización convexa, y aun así las redes llegan a buenos puntos desde inicios razonables
 - c. Demuestra que las redes siempre encuentran mínimos globales
 - d. Significa que las redes no necesitan gradientes
4. ¿Qué es un punto de silla?
- a. Un mínimo global
 - b. Un punto estacionario donde la curvatura de la función cambia, ni mínimo ni máximo
 - c. Un punto donde el gradiente es muy grande
 - d. El punto de partida del descenso de gradiente

Respuestas

1. **b)** Los puntos estacionarios incluyen mínimos, máximos y puntos de silla; para una f genérica el descenso de gradiente solo llega a uno de ellos, según dónde empiece.
2. **a)** Para una función convexa, el descenso de gradiente converge a un mínimo global independientemente de la inicialización.
3. **b)** Cualquier modelo no trivial es no convexo, a diferencia de las SVM; aun así, los modelos diferenciables iniciados con sensatez convergen a puntos que rinden bien.
4. **b)** Los puntos de silla son estacionarios, el gradiente es cero, pero son puntos de inflexión donde cambia la curvatura.

Ejemplos resueltos

Comprobar la desigualdad de convexidad Compruebe la desigualdad de convexidad para $f(x) = x^2$ con $x_1 = 0$, $x_2 = 2$ y $\alpha = 0.5$.

El punto dentro del intervalo es $\alpha x_1 + (1 - \alpha) x_2 = 0.5 \times 0 + 0.5 \times 2 = 1$.

El lado izquierdo es la función ahí: $f(1) = 1$.

El lado derecho es la recta entre los extremos: $0.5 \times f(0) + 0.5 \times f(2) = 0.5 \times 0 + 0.5 \times 4 = 2$.

Como $1 \leq 2$, la función queda por debajo del segmento en este punto, como exige la convexidad.

Respuesta: $1 \leq 2$: la desigualdad se cumple.

Ejercicios

1. Muestre con $x_1 = -1$, $x_2 = 1$ y $\alpha = 0.5$ que $f(x) = -x^2$ no es convexa.

Pista: Calcule ambos lados de la desigualdad de convexidad.

Respuesta: El punto medio es 0, y $f(0) = 0$. El segmento da $0.5 \times f(-1) + 0.5 \times f(1) = 0.5 \times (-1) + 0.5 \times (-1) = -1$. Como 0 no es ≤ -1 , la

desigualdad falla, así que $-x^2$ no es convexa.

2. La función $f(x) = x^3$ tiene $f'(0) = 0$. ¿Es $x = 0$ un mínimo? ¿Qué tipo de punto estacionario es?

Pista: Mire los valores de f justo a la izquierda y justo a la derecha de 0 .

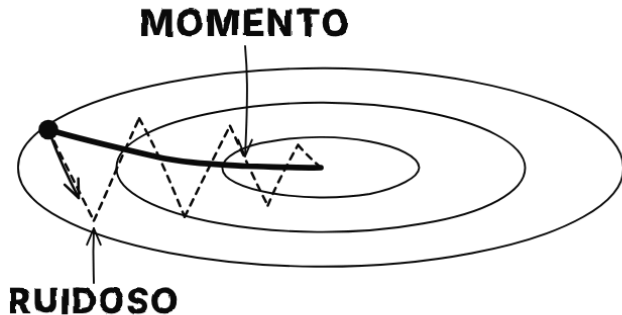
Respuesta: No. f es negativa justo a la izquierda de 0 y positiva justo a la derecha, así que 0 no es ni mínimo ni máximo: es un punto estacionario de inflexión, de tipo silla, donde cambia la curvatura.

Acelerar el descenso de gradiente

El gradiente negativo describe la dirección de descenso más pronunciado, pero solo en un entorno infinitesimalmente pequeño del punto. Como veremos en el capítulo 5, donde presentamos la optimización estocástica, estas direcciones pueden ser extremadamente ruidosas, sobre todo al tratar con modelos grandes. Se han desarrollado diversas técnicas para acelerar la convergencia del algoritmo de optimización seleccionando mejores direcciones de descenso. Por razones computacionales, nos interesan especialmente los métodos que no requieren derivadas de orden superior, como la hessiana, ni múltiples llamadas a la función.

$$\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) + \lambda \mathbf{g}_{t-1}$$

$$\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) - \lambda \eta_t \nabla f(\mathbf{x}_{t-2}) + \lambda^2 \mathbf{g}_{t-2}$$

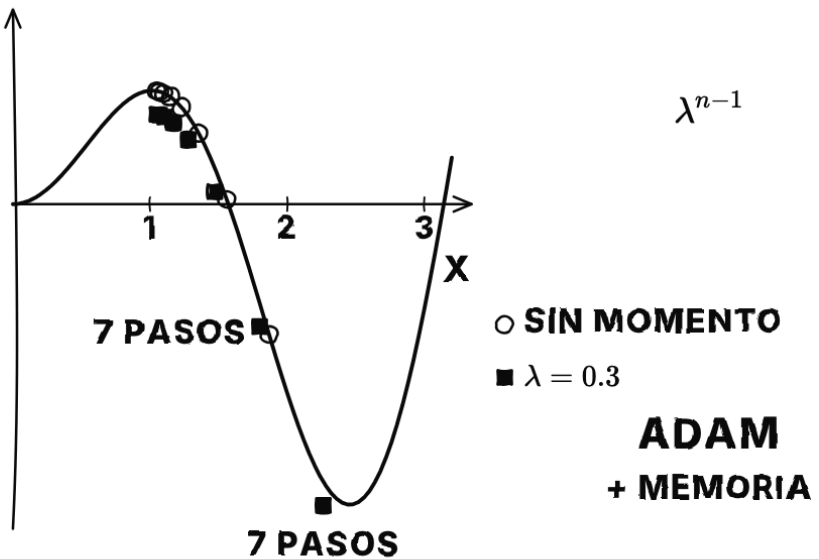


El momento suaviza el camino

Describimos aquí una de estas técnicas, el momento, y remitimos a otras fuentes para una introducción más amplia, como una entrada de blog de 2016 de Sebastian Ruder. Si imagina el descenso de gradiente como una bola que «rueda cuesta abajo», el movimiento es relativamente errático, porque cada gradiente puede apuntar en una dirección completamente distinta; de hecho, para una elección perfecta del tamaño de paso y una función de pérdida convexa, dos gradientes cualesquiera en iteraciones consecutivas serán ortogonales. Podemos suavizar este comportamiento introduciendo un término de «momento» que conserva algo de la dirección de la iteración de gradiente anterior: $\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) + \lambda \mathbf{g}_{t-1}$, el descenso más pronunciado más el término de momento, y luego $\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{g}_t$.

Aquí inicializamos $\mathbf{g}_0 = \mathbf{0}$; la figura F.2.4 muestra un ejemplo. El coeficiente λ determina cuánto se amortigua el término anterior. De hecho, desplegar dos términos da $\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1}) - \lambda \eta_t \nabla f(\mathbf{x}_{t-2}) + \lambda^2 \mathbf{g}_{t-2}$.

La figura compara el descenso de gradiente con y sin momento al minimizar la función $x \sin(2x)$, partiendo de $x = 1 + \varepsilon$, con $\lambda = 0.3$.



$x \sin 2x$, con y sin momento

Generalizando, la iteración en el tiempo $t - n$ se amortigua por un factor λ^{n-1} . Puede demostrarse que el momento acelera el entrenamiento al suavizar el camino de optimización. Otra técnica común es adaptar el tamaño del paso para cada parámetro según la magnitud de los gradientes. Un algoritmo de optimización común que

combina varias de estas ideas es Adam. Una ventaja de Adam es que resulta relativamente robusto a la elección de sus hiperparámetros, siendo un hiperparámetro un parámetro seleccionado por el usuario, en lugar de aprendido por descenso de gradiente, y la elección por defecto en la mayoría de los marcos es un buen punto de partida en la mayoría de los casos. Diseñar nuevos optimizadores que puedan «desbancar» a Adam, o a sus variaciones como AdamW, de su lugar como optimizador por defecto de facto en el aprendizaje profundo sigue siendo un problema de investigación abierto, con trabajos recientes sobre el diseño de optimizadores a medida para redes neuronales desde primeros principios.

Una desventaja de usar algoritmos de optimización acelerados puede ser un mayor requisito de almacenamiento: por ejemplo, el momento nos obliga a guardar en memoria la iteración de gradiente anterior, duplicando el espacio que necesita el algoritmo de optimización, aunque en la mayoría de los casos la memoria necesaria para calcular el gradiente es el factor más influyente, como veremos en la sección 6.3.

De la teoría a la práctica. Un recuadro sobre los ejercicios explica que este libro no tiene ejercicios clásicos de final de capítulo, que ya cubren muchos libros de texto existentes. En su lugar, propongo un camino de autoaprendizaje para ayudarle a explorar dos marcos, JAX y PyTorch, a medida que avanza en el libro. Las soluciones de los ejercicios se publicarán en el sitio web del libro. Estas secciones están llenas de enlaces a material en línea, que podría haber caducado o cambiado de sitio cuando usted lo busque.

Preguntas

1. ¿Qué añade el término de momento al descenso de gradiente simple?
 - a. Una segunda derivada de la función
 - b. Una fracción λ del paso anterior, que suaviza el camino errático
 - c. Una perturbación aleatoria en cada paso
 - d. Solo un tamaño de paso mayor
2. ¿Por qué se describe como errático el camino del descenso de gradiente simple?
 - a. Porque cada gradiente puede apuntar en una dirección completamente distinta de la anterior
 - b. Porque el tamaño del paso es aleatorio
 - c. Porque usa segundas derivadas
 - d. Porque siempre se pasa del mínimo
3. ¿Cuál es un coste de usar momento?
 - a. No puede usarse con funciones convexas
 - b. Necesita la hessiana
 - c. Debe guardar el paso anterior, duplicando la memoria del optimizador
 - d. Necesita varias llamadas a la función por paso
4. ¿Por qué Adam es un optimizador por defecto popular?
 - a. Es relativamente robusto a la elección de sus hiperparámetros, y los valores por defecto suelen funcionar

- b. Encuentra el mínimo global de cualquier función
- c. No necesita gradientes
- d. No usa memoria

Respuestas

1. **b)** El momento conserva algo de la dirección de la iteración anterior, $\lambda \mathbf{g}_{t-1}$, así que la bola que rueda cuesta abajo se mueve de forma menos errática.
2. **a)** Cada gradiente puede apuntar a otro sitio; con un tamaño de paso perfecto sobre una pérdida convexa, los gradientes consecutivos son incluso ortogonales.
3. **c)** La iteración de gradiente anterior debe mantenerse en memoria, duplicando el espacio que necesita el propio algoritmo de optimización.
4. **a)** Adam combina varias ideas y resulta robusto a sus hiperparámetros, con los valores por defecto de los marcos como buen punto de partida.

Ejemplos resueltos

Dos pasos con momento Minimice $f(x) = x^2$ desde $x_0 = 3$ con $\eta = 0.1$, $\lambda = 0.5$ y $g_0 = 0$, usando momento. Compare x_2 con el descenso de gradiente simple, que da 1.92 tras dos pasos.

El gradiente de x^2 es $2x$; las actualizaciones son $\mathbf{g}_t = -\eta \nabla f(\mathbf{x}_{t-1}) + \lambda \mathbf{g}_{t-1}$ y $\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{g}_t$.

Paso 1: el gradiente en 3 es 6, así que $\mathbf{g}_1 = -0.6 + 0.5 \times 0 = -0.6$ y $\mathbf{x}_1 = 3 - 0.6 = 2.4$.

Paso 2: el gradiente en 2.4 es 4.8, así que $\mathbf{g}_2 = -0.48 + 0.5 \times (-0.6) = -0.78$ y $\mathbf{x}_2 = 2.4 - 0.78 = 1.62$.

El momento arrastró parte del primer paso hacia delante, así que tras dos pasos está más cerca del mínimo en 0 que el descenso simple.

Respuesta: $x_1 = 2.4$ y $x_2 = 1.62$ con momento, frente a 1.92 sin él.

Ejercicios

1. Un modelo tiene mil millones de parámetros guardados en 4 bytes cada uno. ¿Cuánta memoria ocupan los parámetros, y cuánta más añade el momento?

Pista: El momento guarda un paso anterior por parámetro.

Respuesta: Los parámetros ocupan 4 mil millones de bytes, unos 4 GB. El momento guarda un valor más por parámetro, así que añade otros 4 GB aproximadamente, duplicando la parte del optimizador.

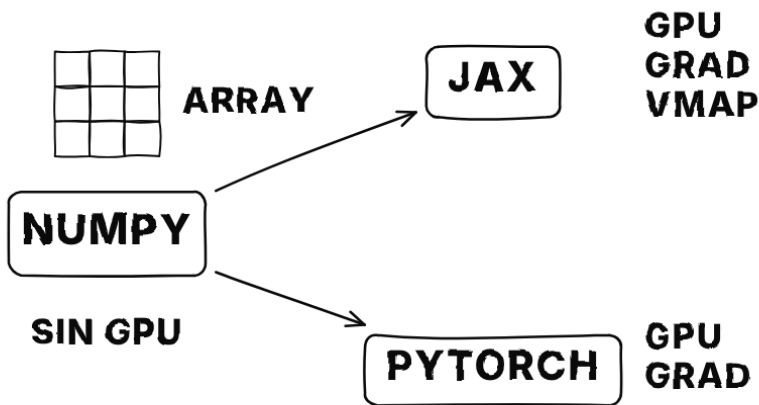
2. Con $\lambda = 0.3$, ¿cuánto vale λ^3 , y qué dice la regla de amortiguación del libro que mide?

Pista: La iteración en el tiempo $t - n$ se amortigua por λ^{n-1} .

Respuesta: $\lambda^3 = 0.027$. Por la regla λ^{n-1} , es el factor de amortiguación de la iteración en el tiempo $t - 4$, que por tanto todavía contribuye, pero solo con alrededor del 3 por ciento de su fuerza.

De la teoría a la práctica, y qué es un conjunto de datos

Empezando por lo básico. El punto de partida para cualquier diseñador de modelos diferenciables es un estudio cuidadoso de NumPy. NumPy implementa un conjunto genérico de funciones para manipular matrices multidimensionales, lo que en el libro llamamos tensores, así como funciones para indexar y transformar su contenido. Puede leer más en la guía de inicio rápido de la biblioteca. Debería sentirse cómodo manejando matrices en NumPy, sobre todo su indexación: el repositorio `numpy-100` de Nicolas Rougier ofrece una serie de ejercicios agradable y pausada para poner a prueba sus conocimientos.

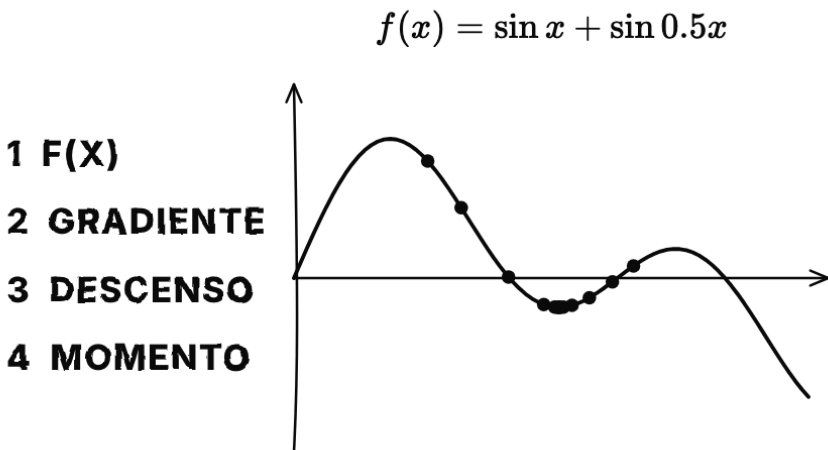


NumPy y sus herederos

Pasar a un marco realista. A pesar de su influencia, NumPy está limitado en su soporte de hardware paralelo como las GPU, a menos que se usen bibliotecas adicionales, y por su falta de diferenciación automática, que se presenta en el capítulo 6. JAX replica la interfaz de NumPy añadiendo un soporte de hardware ampliado, el cálculo automático de gradientes y transformaciones adicionales como el mapeo vectorizado, vmap. Marcos como PyTorch también implementan en su núcleo una interfaz al estilo de NumPy, pero hacen pequeños ajustes de nomenclatura y funcionalidad, y añaden utilidades de alto nivel para construir modelos diferenciables. Tómese su tiempo para hojear la documentación del tipo array en JAX y del tipo tensor en PyTorch, para entender cuánto tienen en común con NumPy. Por ahora, puede ignorar los módulos de alto nivel como el

módulo de redes neuronales de PyTorch. Tendremos más que decir sobre cómo están diseñados estos marcos en el capítulo 6, después de presentar su mecanismo de cálculo de gradientes.

Implementar un algoritmo de descenso de gradiente. Para dominar los tres marcos, NumPy, JAX y PyTorch, sugiero repetir el ejercicio de abajo tres veces; cada variante debería llevar solo unos minutos si conoce la sintaxis. Considere una función bidimensional $f(\mathbf{x})$, con $\mathbf{x} \sim (2)$, donde tomamos como dominio $[0, 10]$: $f(\mathbf{x}) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$. Le pedí a ChatGPT que generara una función agradable con varios mínimos y máximos; nada más en el libro está generado por un modelo de lenguaje, lo que me parece una advertencia cada vez más importante.

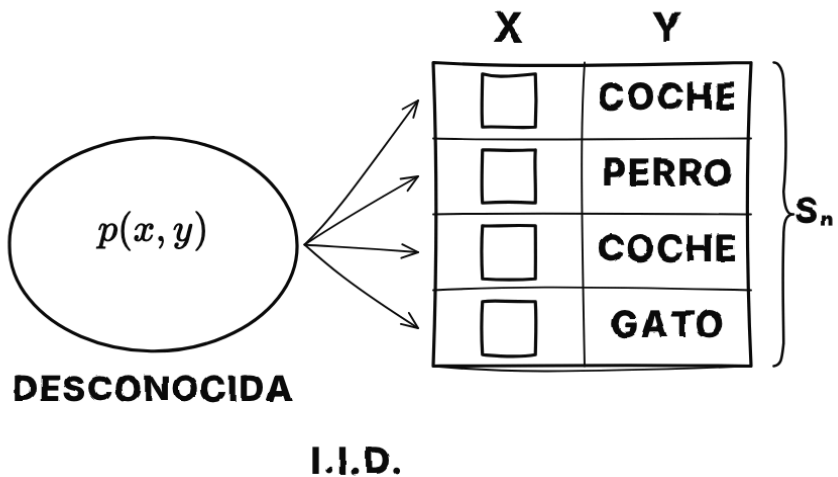


Descenso de gradiente en una función accidentada

Antes de seguir con el libro, repita esto para cada marco. Primero, implemente la función de forma vectorizada: dada una matriz $\mathbf{X} \sim (n, 2)$ de n entradas, debe devolver un vector $f(\mathbf{X}) \sim (n)$ donde $[f(\mathbf{X})]_i = f(\mathbf{X}_i)$. Segundo, implemente otra función para calcular su gradiente, escrito a mano, ya que todavía no hemos tocado la diferenciación automática. Tercero, escriba un procedimiento básico de descenso de gradiente y visualice los caminos que sigue el proceso de optimización desde varios puntos de partida. Cuarto, pruebe a añadir un término de momento y a visualizar la norma de los gradientes, que debería converger a cero a medida que el algoritmo se acerca a un punto estacionario.

Si usa JAX o PyTorch para resolver el ejercicio, el punto tres es un buen lugar para experimentar con `vmap` para vectorizar una función.

Capítulo 3, Conjuntos de datos y pérdidas. En resumen, este capítulo formaliza el escenario del aprendizaje supervisado. Presentamos los conceptos de conjuntos de datos, pérdidas, minimización del riesgo empírico y los supuestos básicos que se hacen en el aprendizaje supervisado. Cerramos ofreciendo una formulación probabilística del aprendizaje supervisado construida sobre la noción de máxima verosimilitud. Este breve capítulo sirve de columna vertebral para el resto del libro.



Un conjunto de datos de pares

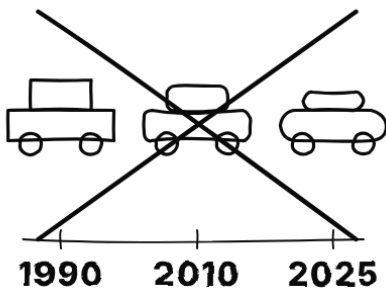
Sección 3.1, ¿Qué es un conjunto de datos? Consideramos un escenario en el que programar a mano cierta función es inviable, por ejemplo reconocer objetos en imágenes del mundo real, pero reunir ejemplos del comportamiento deseado es suficientemente fácil. Abundan los ejemplos de esto, desde el reconocimiento del habla hasta la navegación de robots. Formalizamos esta idea con la siguiente definición.

Definición, conjunto de datos: un conjunto de datos supervisado $\mathcal{S}_n$ de tamaño n es un conjunto de n pares, $\mathcal{S}_n = (x_1, y_1), \dots, (x_n, y_n)$, donde cada (x_i, y_i) es un ejemplo de una relación entrada-salida que queremos modelar. Suponemos además que cada ejemplo es una

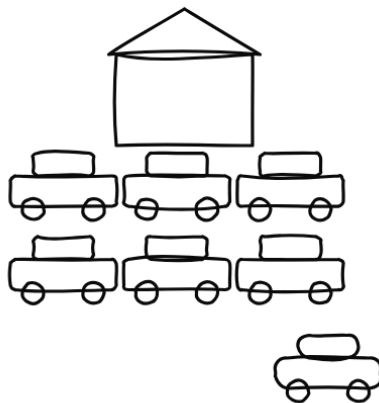
extracción independiente e idénticamente distribuida, i.i.d., de alguna distribución de probabilidad desconocida, e incognoscible, $p(x, y)$.

Consulte el apéndice si, al leer la definición, quiere repasar la teoría de la probabilidad. El último supuesto parece técnico, pero está ahí para asegurar que la relación que intentamos modelar tenga sentido. En particular, que las muestras estén idénticamente distribuidas significa que intentamos aproximar algo suficientemente estable e invariable en el tiempo. Como ejemplo representativo, considere la tarea de reunir un conjunto de datos para reconocer modelos de coches a partir de fotos. Este supuesto se cumplirá si recogemos imágenes durante un periodo corto, pero no será válido si recogemos imágenes de las últimas décadas, ya que los modelos de coches habrán cambiado con el tiempo. En este último caso, entrenar y desplegar un modelo con este conjunto de datos fracasará, pues no podrá reconocer los modelos nuevos o tendrá un rendimiento inferior al óptimo cuando se use.

CON EL TIEMPO



UN CONCESION.

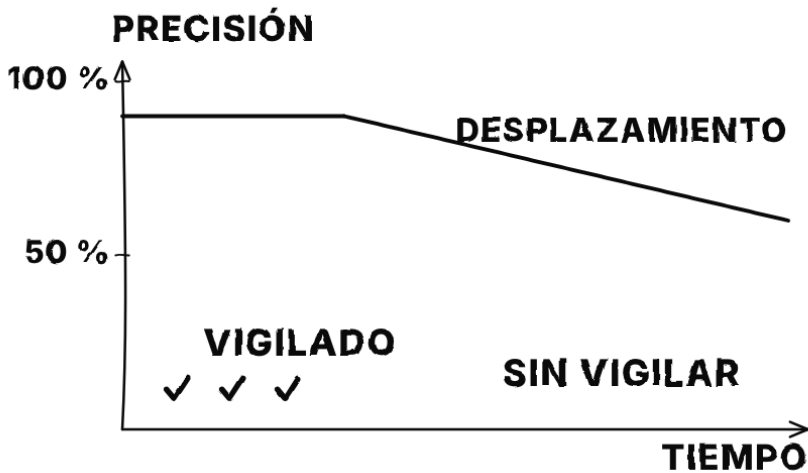


ROMA ✓ TOKIO ✗

Cuando i.i.d. se rompe

De modo similar, que las muestras estén independientemente distribuidas significa que nuestro conjunto de datos no tiene sesgo en su recogida, y que es suficientemente representativo de toda la distribución. Volviendo al ejemplo anterior, reunir imágenes cerca de un concesionario de Tesla no será válido, ya que recogeremos una sobreabundancia de imágenes de cierto tipo mientras perdemos imágenes de otras marcas y modelos. Observe que la validez de estos supuestos depende del contexto: un conjunto de datos de coches recogido en Italia puede ser válido al desplegar nuestro modelo en Roma o Milán, mientras que puede no serlo al desplegarlo en Tokio o en Taiwán. El supuesto i.i.d. siempre debe comprobarse con cuidado, para asegurar que aplicamos nuestras herramientas de aprendizaje supervisado a un escenario válido. Curiosamente, los LLM modernos

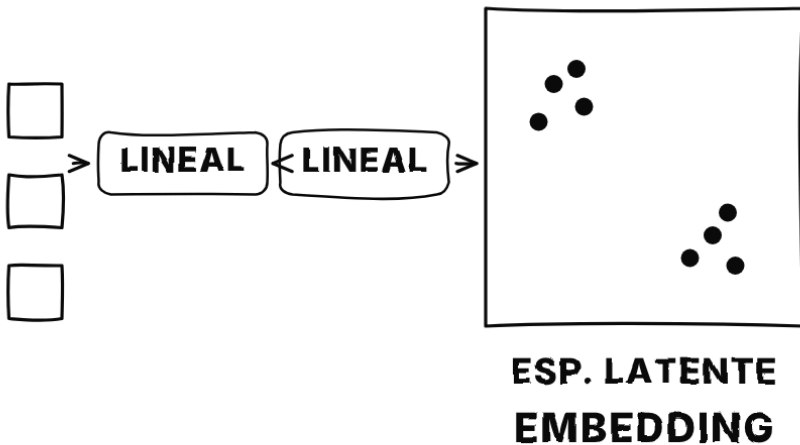
se entrenan con distribuciones de datos tan grandes que incluso entender qué tareas están realmente dentro de la distribución y cuáles fuera de ella, y cuánto son capaces de generalizar los modelos, se vuelve borroso.



Desplazamiento de dominio con el tiempo

Un recuadro añade más sobre la propiedad i.i.d.. Es importante notar que asegurar la propiedad i.i.d. no es un proceso de una sola vez, y debe comprobarse constantemente durante la vida de un modelo. En el caso de la clasificación de coches, si no se vigila, cambios sutiles en la distribución de los coches a lo largo del tiempo degradarán el rendimiento de un modelo de aprendizaje automático, un ejemplo de desplazamiento de dominio. Como otro ejemplo, un sistema de recomendación cambiará la forma en que los usuarios interactúan con

cierta aplicación, ya que empezarán a reaccionar a las sugerencias del propio sistema de recomendación. Esto crea bucles de retroalimentación que exigen una reevaluación constante del rendimiento del sistema y de la aplicación.



Bucles y embeddings

Una figura muestra que los modelos diferenciables procesan los datos transformándolos secuencialmente mediante operaciones de álgebra lineal. En muchos casos, después de optimizar estos programas, las representaciones internas de los datos de entrada en el modelo, lo que llamamos un modelo preentrenado, tienen propiedades geométricas: por ejemplo, imágenes semánticamente similares se proyectan a puntos cercanos en este espacio «latente». Transformar los datos de

un espacio no métrico, las imágenes de entrada originales, a un espacio métrico se llama hacer un embedding de los datos.

Preguntas

1. ¿Qué añade JAX sobre la interfaz de NumPy?
 - a. Nada; es una copia con otro nombre
 - b. Soporte de hardware ampliado, gradientes automáticos y transformaciones como vmap
 - c. Solo un módulo de redes neuronales
 - d. Una convención de indexación distinta que empieza en uno
2. ¿Por qué un conjunto de fotos de coches reunido a lo largo de varias décadas rompería el supuesto i.i.d.?
 - a. Porque las fotos antiguas tienen menor resolución
 - b. Porque los modelos de coches cambian con el tiempo, así que las muestras no están idénticamente distribuidas
 - c. Porque habría demasiadas fotos
 - d. Porque las fotos no serían independientes en cuanto a la iluminación de unas y otras
3. ¿Por qué es un problema recoger fotos cerca de un concesionario de Tesla?
 - a. Sesga la recogida hacia un tipo de coche, así que las muestras no son representativas: falla la parte de independencia
 - b. Los Tesla son difíciles de reconocer
 - c. Rompe solo la parte de idénticamente distribuidas

- d. Hace que el conjunto de datos sea demasiado pequeño
- 4. ¿Por qué debe comprobarse la propiedad i.i.d. durante toda la vida de un modelo?
 - a. Porque los modelos olvidan su entrenamiento
 - b. Porque las distribuciones derivan, por ejemplo por desplazamiento de dominio o bucles de retroalimentación como un recomendador que cambia el comportamiento de los usuarios
 - c. Porque los marcos cambian sus valores por defecto
 - d. Porque los conjuntos de datos se borran tras el entrenamiento

Respuestas

- 1. **b)** JAX replica la interfaz de NumPy añadiendo soporte de hardware, cálculo automático de gradientes y transformaciones como el mapeo vectorizado.
- 2. **b)** Idénticamente distribuidas significa que lo que se aprende es estable en el tiempo; los modelos de coches cambian a lo largo de las décadas.
- 3. **a)** Independientemente distribuidas significa que no hay sesgo en la recogida; un solo concesionario da una sobreabundancia de una marca.
- 4. **b)** Cambios sutiles en los datos con el tiempo degradan el rendimiento, y un sistema de recomendación incluso cambia el comportamiento del que aprende.

Ejemplos resueltos

Un paso de gradiente en la función de práctica del libro Para $f(x) = \sin(x_1)\cos(x_2) + \sin(0.5x_1)\cos(0.5x_2)$, calcule el gradiente en $(\pi,$

o), dé un paso de descenso de gradiente con $\eta = 0.1$ y compruebe que f disminuye.

Las derivadas parciales son $\partial f / \partial x_1 = \cos(x_1)\cos(x_2) + 0.5\cos(0.5x_1)\cos(0.5x_2)$ y $\partial f / \partial x_2 = -\sin(x_1)\sin(x_2) - 0.5\sin(0.5x_1)\sin(0.5x_2)$.

En $(\pi, 0)$: $\partial f / \partial x_1 = (-1)(1) + 0.5 \times \cos(\pi/2) \times 1 = -1$, y $\partial f / \partial x_2 = 0$, porque $\sin(0) = 0$; así que el gradiente es $(-1, 0)$.

El paso es $x - \eta \nabla f = (\pi, 0) - 0.1 \times (-1, 0) = (\pi + 0.1, 0)$, aproximadamente $(3.2416, 0)$.

Comprobación: $f(\pi, 0) = 0 + \sin(\pi/2) = 1$, mientras que $f(\pi + 0.1, 0) = \sin(\pi + 0.1) + \sin(\pi/2 + 0.05) \approx -0.0998 + 0.9988 = 0.8990$, que es menor.

Respuesta: Gradiente $(-1, 0)$; nuevo punto aproximadamente $(3.2416, 0)$; f baja de 1 a unos 0.899.

Ejercicios

1. Su f vectorizada recibe una matriz X de forma $(5, 2)$. ¿Qué forma debe tener la salida, y cuál es su tercer elemento?

Pista: El libro pide $[f(X)]_i = f(X_i)$.

Respuesta: Un vector de forma (5) ; su tercer elemento es f aplicada a la tercera fila de X .

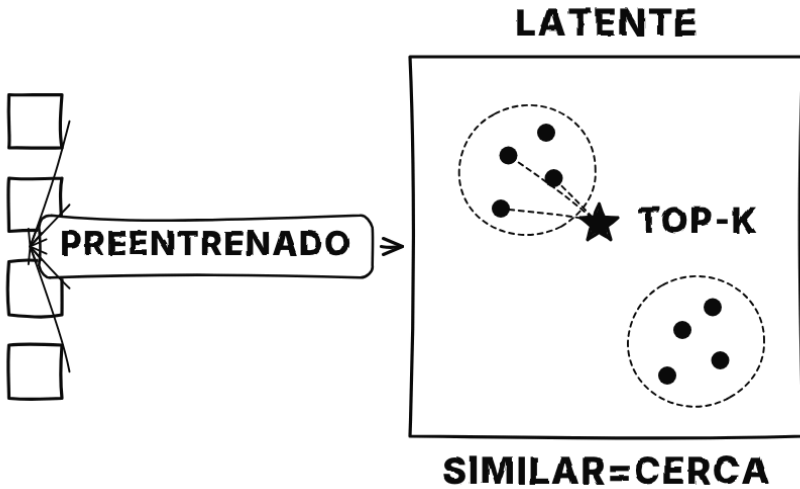
2. Un filtro de spam entrenado con correos de 2015 se despliega hoy. Diga qué parte del supuesto i.i.d. corre más riesgo y por qué.

Pista: Pregúntese si lo que se aprende ha seguido siendo lo mismo con el tiempo.

Respuesta: La parte de idénticamente distribuidas: los estilos y temas del spam derivan a lo largo de los años, así que los correos de hoy vienen de una distribución distinta de la de los datos de entrenamiento, un caso de desplazamiento de dominio.

Variantes del aprendizaje supervisado

Existen muchas variaciones del escenario estándar de aprendizaje supervisado, aunque la mayoría de las aplicaciones exitosas usan el aprendizaje supervisado de una forma u otra. Por ejemplo, algunos conjuntos de datos pueden no tener disponibles los objetivos y_i , en cuyo caso hablamos de aprendizaje no supervisado. Aplicaciones típicas del aprendizaje no supervisado son los algoritmos de agrupamiento, en los que queremos agregar nuestros datos de entrada en grupos tales que los puntos de un grupo sean similares y los puntos de grupos distintos sean disímiles. Como otro ejemplo, en un sistema de recuperación podemos querer buscar en una gran base de datos los k elementos más similares a una consulta dada por un usuario.



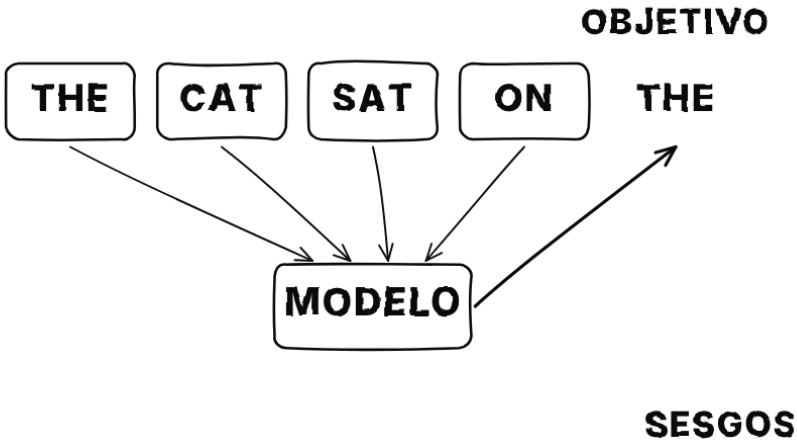
Grupos en un espacio latente

Al tratar con datos complejos como imágenes, esto no es trivial, porque las distancias entre imágenes están mal definidas si operamos sobre píxeles: incluso perturbaciones pequeñas pueden modificar millones de píxeles. Sin embargo, suponga que tenemos disponible algún modelo diferenciable que ya hemos optimizado para alguna otra tarea que suponemos suficientemente genérica, por ejemplo la clasificación de imágenes. Lo llamamos un modelo preentrenado. Como veremos, los estados internos de este modelo pueden interpretarse como vectores en un espacio de alta dimensión. En muchos casos, se ha mostrado que estos vectores tienen propiedades geométricas útiles, en el sentido de que los objetos semánticamente similares se envían, se incrustan, en puntos cercanos de estas representaciones. Por tanto, podemos usar estas representaciones

latentes con modelos de agrupamiento estándar, como los modelos de mezcla de gaussianas. La figura F.3.1 da una visión general de esta idea.

¿Y si no tenemos acceso a un modelo preentrenado? Una variación común del aprendizaje no supervisado se llama aprendizaje autosupervisado, SSL. El objetivo del aprendizaje autosupervisado es encontrar automáticamente algún objetivo supervisado en un conjunto de datos no supervisado genérico, para optimizar un modelo que pueda usarse en un gran conjunto de tareas posteriores. Por ejemplo, si tenemos acceso a un gran corpus de texto, siempre podemos optimizar un programa para predecir cómo es probable que continúe un pequeño fragmento de texto. La constatación de que las redes neuronales también pueden realizar un embedding eficiente de texto cuando se preentrenan de forma autosupervisada tuvo un profundo impacto en la comunidad. Los grandes conjuntos de datos de la web también están llenos de sesgos, groserías y contenido vulgar; reconocer que los modelos entrenados con estos datos internalizan esos sesgos fue otra constatación importante, y es una de las principales críticas a los modelos fundacionales de código cerrado.

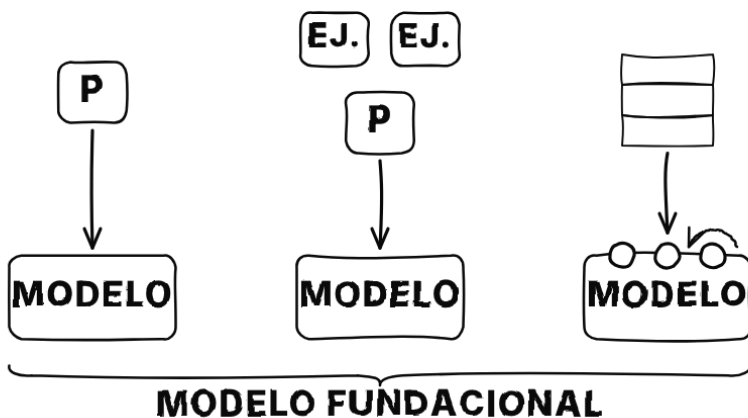
AUTOSUPERVIS.



Aprender del propio texto

La figura F.3.2 muestra tres formas de usar modelos entrenados. Sin ejemplos: se da directamente una pregunta al modelo, lo que puede lograrse con modelos de lenguaje generativos, presentados en el capítulo 8. El prompting con pocos ejemplos es similar, pero se proporcionan unos pocos ejemplos como entrada. Ambas técnicas solo pueden emplearse si el modelo subyacente muestra una gran capacidad de generalización. Ajuste fino: el modelo se optimiza mediante descenso de gradiente sobre un pequeño conjunto de ejemplos, lo que procede de forma similar a entrenar el modelo desde cero.

ZERO-SHOT FEW-SHOT AJUSTE FINO

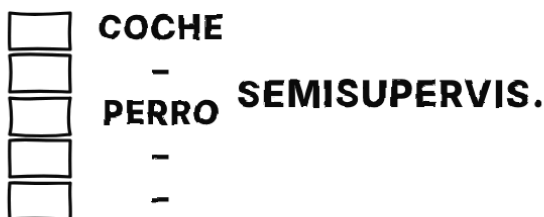


Sin ejemplos, pocos ejemplos, ajuste fino

Como veremos en los capítulos 8 y 10, los LLM pueden verse como iteraciones modernas de esta idea básica, ya que optimizar modelos como GPT o Llama siempre empieza con un entrenamiento autosupervisado básico en forma de predicción del siguiente token. Estos modelos a veces se llaman modelos fundacionales. En el caso más simple, pueden usarse tal cual para una tarea nueva, como responder a una consulta: en este caso, decimos que se usan sin ejemplos. Para los LLM, también es posible proporcionar un pequeño número de ejemplos de una tarea nueva como instrucción de entrada, en cuyo caso hablamos de prompting con pocos ejemplos. En el caso más general, podemos tomar un modelo fundacional preentrenado y optimizar sus parámetros por descenso de gradiente en una tarea nueva: esto se llama hacer un ajuste fino del modelo. La figura F.3.2

compara los tres enfoques. En este libro nos centramos en construir modelos desde cero, pero el ajuste fino puede hacerse por medios similares.

El ajuste fino resulta especialmente fácil gracias a la presencia de grandes repositorios de código abierto en línea. El ajuste fino puede hacerse sobre el conjunto completo de parámetros del modelo de partida, o considerando solo un subconjunto más pequeño o un pequeño número de parámetros adicionales: esto se llama ajuste fino eficiente en parámetros, PEFT. El aprendizaje con pocos ejemplos también puede hacerse ajustando el modelo; en los casos en que el ajuste fino no es necesario, decimos que el modelo realiza aprendizaje en contexto. Consideraremos las técnicas de ajuste fino eficiente en parámetros en el próximo volumen.



Ajustar pocos parámetros

Son posibles muchas otras variaciones del aprendizaje supervisado, que no tenemos espacio para enumerar aquí en detalle, salvo algunas pistas genéricas. Si solo partes de un conjunto de datos están etiquetadas, tenemos un escenario semisupervisado; veremos algunos ejemplos de aprendizaje semisupervisado en el capítulo 12. Además, podemos tener escenarios con varios conjuntos de datos pertenecientes a distribuciones «similares», o a la misma distribución en distintos periodos de tiempo, lo que da lugar a incontables problemas según el orden en que se proporcionan las tareas o los datos, como la adaptación de dominio, el metaaprendizaje, el aprendizaje continuo, el aprendizaje de métricas, el desaprendizaje, y así sucesivamente. Algunos de estos se tratarán en el próximo volumen.

Preguntas

1. ¿Por qué no podemos simplemente agrupar imágenes comparando sus píxeles?
 - a. Porque los píxeles no son números
 - b. Porque las distancias sobre píxeles están mal definidas: perturbaciones pequeñas pueden cambiar millones de píxeles
 - c. Porque las imágenes tienen muy pocos píxeles
 - d. Porque el agrupamiento solo funciona con texto
2. ¿Qué hace posible el aprendizaje autosupervisado sobre un corpus de texto sin etiquetar?
 - a. Personas etiquetan cada frase
 - b. Se encuentra automáticamente un objetivo supervisado en los datos, como predecir cómo continúa un fragmento de texto
 - c. Se le dan al modelo las respuestas de antemano
 - d. El agrupamiento sustituye al entrenamiento
3. ¿Qué distingue el ajuste fino del prompting con pocos ejemplos?
 - a. El ajuste fino cambia los parámetros del modelo por descenso de gradiente; el prompting con pocos ejemplos solo pone ejemplos en la entrada
 - b. El prompting con pocos ejemplos necesita más datos
 - c. El ajuste fino no necesita ejemplos
 - d. Son lo mismo

4. ¿Qué es un escenario semisupervisado?

- a. Uno sin ningún dato
- b. Uno donde solo partes del conjunto de datos están etiquetadas
- c. Uno con dos modelos
- d. Uno donde las etiquetas cambian con el tiempo

Respuestas

- 1. **b)** Las distancias entre píxeles no reflejan el significado; el libro usa en su lugar los vectores internos de un modelo preentrenado, donde los objetos similares caen cerca unos de otros.
- 2. **b)** El SSL encuentra un objetivo supervisado dentro de datos no supervisados, por ejemplo la predicción del siguiente token, como hacen los LLM.
- 3. **a)** El prompting con pocos ejemplos proporciona ejemplos en la instrucción, en contexto; el ajuste fino optimiza los parámetros sobre un conjunto de datos pequeño.
- 4. **b)** Si solo parte de un conjunto de datos tiene etiquetas, el escenario es semisupervisado.

Ejercicios

1. Quiere encontrar, entre un millón de fotos de vacaciones, las diez más similares a una nueva. Usando el pasaje, describa un enfoque viable.

Pista: No compare píxeles; piense en un modelo preentrenado.

Respuesta: Pase cada foto por un modelo preentrenado, como un clasificador de imágenes, y tome su vector interno para cada foto. Las fotos semánticamente similares quedan incrustadas cerca unas de otras,

así que recupera las diez fotos cuyos vectores están más cerca del vector de la foto nueva, una recuperación top-k en el espacio latente.

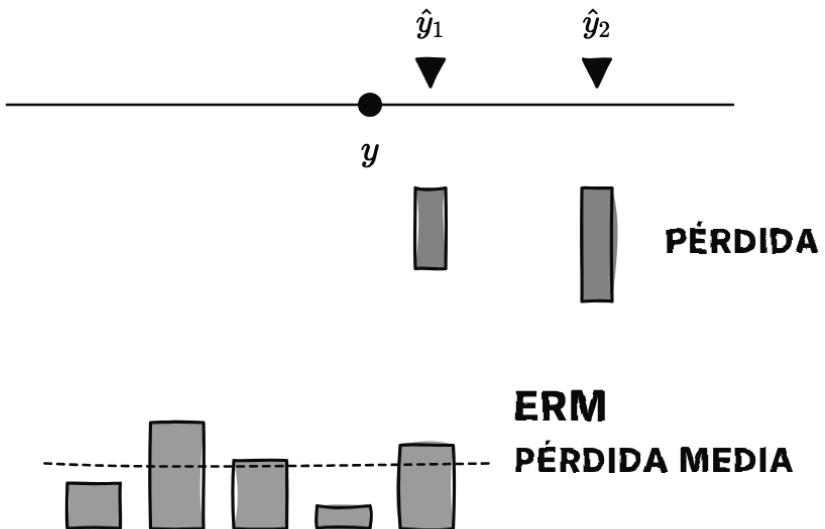
2. Nombre las tres formas de usar un modelo entrenado de la figura F.3.2 y diga cuáles cambian sus parámetros.

Pista: Dos solo cambian la entrada.

Respuesta: Sin ejemplos, donde la pregunta se da directamente; prompting con pocos ejemplos, donde se añaden unos pocos ejemplos a la entrada; y ajuste fino, donde el modelo se optimiza por descenso de gradiente sobre un conjunto de datos pequeño. Solo el ajuste fino cambia los parámetros.

Funciones de pérdida

Una vez reunidos los datos, necesitamos formalizar nuestra idea de «aproximar» el comportamiento deseado, lo que hacemos introduciendo el concepto de funciones de pérdida.



Una pérdida ordena las predicciones

Definición, función de pérdida: dado un objetivo deseado y y el valor predicho $\hat{y} = f(x)$ de un modelo f , una función de pérdida $l(y, \hat{y}) \in \mathbb{R}$ es una función escalar y diferenciable cuyo valor se correlaciona con el rendimiento del modelo. Es decir, $l(y, \hat{y}_1) < l(y, \hat{y}_2)$ significa que la

predicción $\hat{y}_1$ es mejor que la predicción $\hat{y}_2$ al considerar el valor de referencia, el objetivo y .

Una función de pérdida incorpora nuestra comprensión de la tarea y nuestras preferencias en el espacio de soluciones en una escala de valores reales que puede aprovecharse en un algoritmo de optimización. Al ser diferenciable, nos permite convertir nuestro problema de aprendizaje en un problema de optimización matemática que puede resolverse mediante descenso de gradiente minimizando la pérdida media sobre nuestro conjunto de datos.

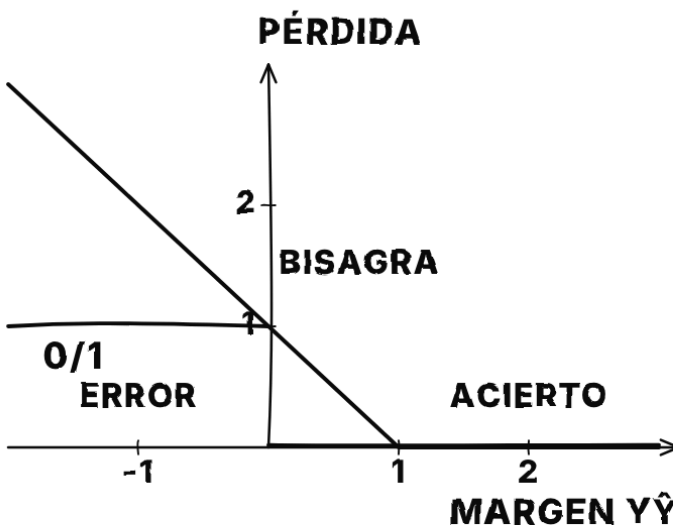
Con este fin, dado un conjunto de datos $\mathcal{S}_n$ de pares (x_i, y_i) y una función de pérdida l , una tarea de optimización sensata es la mínima pérdida media sobre el conjunto de datos alcanzable por cualquier modelo diferenciable posible f : $f^* = \operatorname{argmin}_f (1/n) \sum_{i=1}^n l(y_i, f(x_i))$, el promedio sobre el conjunto de datos de la pérdida de la predicción en la muestra i -ésima.

Por razones históricas, la ecuación E.3.1 se denomina minimización del riesgo empírico, ERM, donde riesgo se usa como sinónimo genérico de pérdida. El recuadro que sigue dice más sobre el origen del término.

En la ecuación E.3.1 estamos suponiendo implícitamente que minimizamos en el espacio de todas las funciones posibles definidas sobre nuestra entrada x . Veremos en breve que nuestros modelos siempre pueden parametrizarse mediante un conjunto de tensores w , llamados los parámetros del modelo, y la minimización se hace buscando el valor óptimo de estos parámetros mediante optimización numérica, lo que denotamos por $f(x, w)$. Por tanto, dado un conjunto de datos $\mathcal{S}_n$, una función de pérdida l y un espacio de modelos f , podemos entrenar nuestro modelo optimizando el riesgo empírico de

la ecuación E.3.1 mediante descenso de gradiente, ecuación E.2.27: $w^* = \operatorname{argmin}_w (1/n) \sum_{i=1}^n l(y_i, f(x_i, w))$, donde la minimización se hace ahora respecto al tensor de parámetros w .

Sobre la diferenciabilidad de la pérdida. Antes de continuar, hacemos dos observaciones sobre el marco ERM. Primero, observe que el requisito de diferenciabilidad de l es fundamental. Considere una tarea simple de clasificación binaria, que presentaremos como corresponde en el próximo capítulo, donde y solo puede tomar dos valores, -1 o $+1$. Dado un modelo de valor real $f(x) \in \mathbb{R}$, podemos identificar las dos decisiones con el signo de f , que denotamos $\operatorname{sign}(f(x))$, y definir una pérdida 0/1 como sigue: $l(y, \hat{y})$ es 0 si $\operatorname{sign}(\hat{y}) = y$, y 1 en otro caso.



Pérdida 0/1 frente a pérdida de bisagra

Aunque esto encaja con nuestra noción intuitiva de «acertar», es inútil como función de pérdida, ya que su gradiente será casi siempre cero, salvo donde cambia el signo de f , y cualquier algoritmo de descenso de gradiente se quedará atascado en la inicialización. Una cantidad menos intuitiva en este caso es el margen $y\hat{y}$, que es positivo o negativo según el signo del modelo coincida, o no, con el deseado, pero que varía de forma continua, a diferencia de la pérdida 0/1 de la ecuación E.3.3. Una posible función de pérdida en este caso es la pérdida de bisagra $l(y, \hat{y}) = \max(0, 1 - y\hat{y})$, que se usa para entrenar modelos de vectores de soporte. Dejando los detalles aparte, esto muestra la tensión inherente entre diseñar funciones de pérdida que codifiquen nuestra noción de rendimiento y que al mismo tiempo sean útiles para la optimización numérica.

Un recuadro sobre riesgo y pérdida: la minimización del riesgo empírico y esperado planteada de este modo se asocia generalmente con el trabajo del informático ruso Vladimir Vapnik, que dio origen al campo de la teoría del aprendizaje estadístico, SLT. La teoría del aprendizaje estadístico se ocupa especialmente del comportamiento de la ecuación E.3.1 vista como una aproximación con muestra finita de la ecuación E.3.5, bajo alguna clase restringida de funciones f y alguna medida de la complejidad subyacente. Las propiedades contraintuitivas de las redes neuronales modernas, como una fuerte generalización mucho después de cuando debería haberse esperado el sobreajuste, han abierto muchas nuevas vías de investigación en la teoría del aprendizaje estadístico; véase también la introducción del capítulo 9.

Preguntas

1. ¿Por qué la pérdida 0/1 es inútil para el descenso de gradiente?
 - a. Porque es demasiado costosa de calcular
 - b. Porque su gradiente es casi siempre cero, así que el descenso de gradiente se queda atascado en la inicialización
 - c. Porque puede ser negativa
 - d. Porque solo funciona para regresión
2. ¿Qué minimiza la minimización del riesgo empírico?
 - a. La mayor pérdida sobre el conjunto de datos
 - b. La pérdida media de las predicciones del modelo sobre el conjunto de datos
 - c. El número de parámetros
 - d. La pérdida sobre datos futuros no vistos, directamente
3. ¿Qué es el margen $\hat{y}$ en la clasificación binaria?
 - a. Una cantidad que es positiva cuando el signo del modelo coincide con el objetivo y negativa en otro caso, y que varía de forma continua
 - b. La pérdida 0/1 multiplicada por dos
 - c. La distancia entre dos clases en píxeles
 - d. Siempre un número positivo

Respuestas

1. **b)** La pérdida 0/1 es plana salvo donde cambia el signo de f , así que su gradiente no da ninguna dirección hacia la que moverse.

2. b) ERM busca el modelo, en la práctica los parámetros w , que minimiza el promedio de $l(y_i, f(x_i))$ sobre los n ejemplos.

3. a) El margen es positivo o negativo según coincidan o no los signos, y a diferencia de la pérdida 0/1 cambia de forma continua.

Ejemplos resueltos

Pérdida de bisagra frente a pérdida 0/1 Calcule la pérdida de bisagra $\max(0, 1 - \hat{y})$ y la pérdida 0/1 para tres predicciones: $y = +1$ con $\hat{y} = 0.3$; $y = +1$ con $\hat{y} = 2$; $y = -1$ con $\hat{y} = 0.5$. Luego calcule la pérdida de bisagra media.

Primer caso: el margen es $1 \times 0.3 = 0.3$, así que la pérdida de bisagra es $\max(0, 0.7) = 0.7$; el signo es correcto, así que la pérdida 0/1 es 0.

Segundo caso: el margen es 2, así que la pérdida de bisagra es $\max(0, -1) = 0$; la pérdida 0/1 es 0.

Tercer caso: el margen es $-1 \times 0.5 = -0.5$, así que la pérdida de bisagra es $\max(0, 1.5) = 1.5$; el signo es incorrecto, así que la pérdida 0/1 es 1.

El riesgo empírico bajo la pérdida de bisagra es el promedio: $(0.7 + 0 + 1.5) / 3 = 2.2 / 3 \approx 0.733$.

Respuesta: Pérdidas de bisagra 0.7, 0 y 1.5 (promedio ≈ 0.733); pérdidas 0/1 de 0, 0 y 1.

Ejercicios

1. En el primer caso de arriba, la predicción pasa de $\hat{y} = 0.3$ a $\hat{y} = 0.5$. ¿Cómo cambian la pérdida de bisagra y la pérdida 0/1, y qué dice eso sobre sus gradientes?

Pista: Vuelva a calcular ambas pérdidas.

Respuesta: La pérdida de bisagra baja de 0.7 a 0.5, mientras que la pérdida 0/1 sigue en 0. La pérdida de bisagra reacciona al cambio, así que tiene un gradiente útil; la pérdida 0/1 es plana ahí, con gradiente cero.

2. Explique con sus propias palabras por qué riesgo se usa como sinónimo de pérdida en la minimización del riesgo empírico, y qué tiene de empírico.

Pista: Piense sobre qué se toma el promedio.

Respuesta: Riesgo es un nombre histórico para la pérdida esperada. Es empírico porque el promedio se toma sobre el conjunto de datos finito que realmente tenemos, una muestra de ejemplos, y no sobre toda la distribución desconocida de los datos.

Riesgo esperado y elección de una pérdida

Como segunda observación, note que el riesgo empírico siempre es trivial de minimizar, definiendo $f(x)$ como sigue: devuelve y si el par (x, y) está en el conjunto de entrenamiento S_n , y un valor por defecto $\bar{y}$, por ejemplo 0, en otro caso.

MEMORIZADO

X	Y
x_1	y_1
x_2	y_2
x_3	y_3

PÉRDIDA 0

$x_{new} \rightarrow \boxed{0}$ **PÉRDIDA ALTA**

¿APRENDIDO?

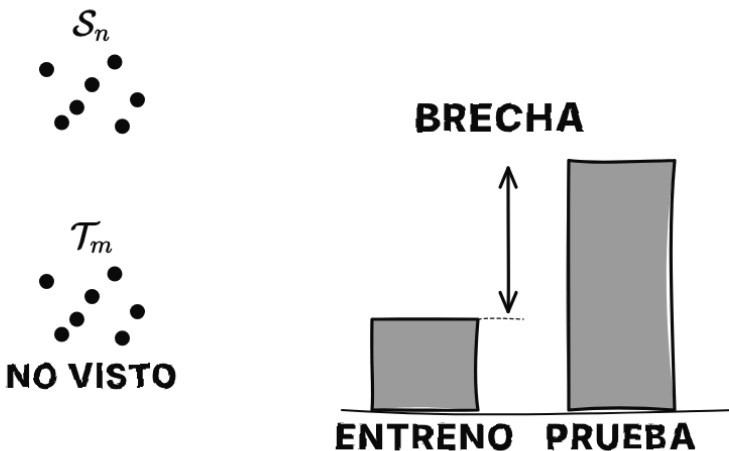
Una tabla de consulta hace trampa

Esta es una tabla de consulta que devuelve una predicción y si el par (x, y) está contenido en el conjunto de datos, y en otro caso recurre a alguna predicción constante $\bar{y}$, por ejemplo 0. Suponiendo que la

pérdida alcanza su cota inferior siempre que $y = \hat{y}$, este modelo siempre logrará el menor valor posible de riesgo empírico, sin aportar ningún valor práctico real.

Esto muestra la diferencia entre memorización y aprendizaje, u optimización. Aunque busquemos un modelo optimizando alguna cantidad de pérdida media sobre nuestros datos de entrenamiento, como en la ecuación E.3.1, nuestro verdadero objetivo es minimizar esta cantidad sobre alguna entrada futura desconocida, aún por ver. Los elementos de nuestro conjunto de entrenamiento son solo un sustituto para este fin. Podemos formalizar esta idea definiendo el problema de minimización del riesgo esperado.

$$ER[f] = \mathbb{E} [l(y, f(x))]$$



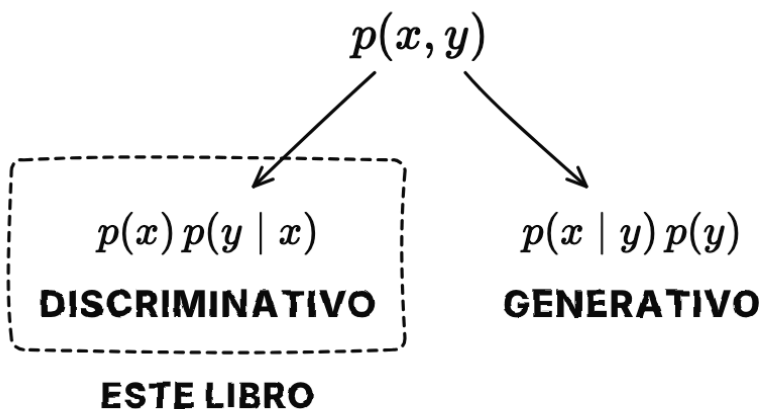
La brecha de generalización

Definición, riesgo esperado: dada una distribución de probabilidad $p(x, y)$ y una función de pérdida l , el riesgo esperado, ER , se define como $ER[f] = \mathbb{E}_{p(x,y)}[l(y, f(x))]$.

Minimizar la ecuación E.3.5 puede interpretarse como minimizar la pérdida media, o esperada, sobre todos los pares entrada-salida posibles, por ejemplo todos los correos posibles, que nuestro modelo podría ver. Claramente, un modelo con bajo riesgo esperado tendría garantizado funcionar correctamente. Sin embargo, la cantidad de la ecuación E.3.5 es inviable de calcular en la práctica, ya que enumerar y etiquetar todos los puntos de datos es imposible. El riesgo empírico proporciona una estimación del riesgo esperado bajo la elección de un conjunto de datos dado, y puede verse como una aproximación de Monte Carlo del riesgo esperado.

La diferencia de pérdida entre el riesgo esperado y el empírico se llama brecha de generalización: un algoritmo de pura memorización como el de la ecuación E.3.4 generalizará mal o, en otros términos, se sobreajustará a los datos de entrenamiento concretos que le dimos. La generalización puede comprobarse en la práctica manteniendo un conjunto de datos de prueba separado $\mathcal{T}_m$ con m puntos de datos nunca usados durante el entrenamiento, de modo que $\mathcal{S}_n \cap \mathcal{T}_m = \emptyset$. Entonces la diferencia de pérdida empírica entre $\mathcal{S}_n$ y $\mathcal{T}_m$ puede usarse como medida aproximada del sobreajuste.

¿Cómo seleccionar una función de pérdida válida? Si no lo ha hecho ya, este es un buen momento para estudiar, u hojear, el material del apéndice, especialmente las distribuciones de probabilidad, los estadísticos suficientes y la estimación de máxima verosimilitud.



Dos formas de dividir $p(x, y)$

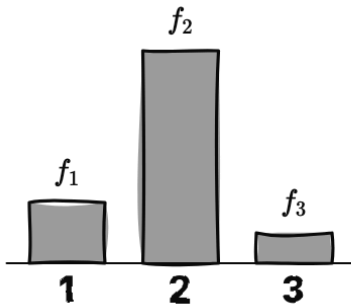
Como veremos en los próximos capítulos, la pérdida codifica nuestro conocimiento a priori de la tarea a resolver, y tiene un gran impacto en el rendimiento. En algunos casos, consideraciones simples sobre el problema bastan para diseñar pérdidas válidas, como se hizo con la pérdida de bisagra en la sección anterior.

Sin embargo, es posible trabajar de forma más fundamentada reformulando todo el proceso de entrenamiento en términos puramente probabilísticos, como mostramos ahora. Esta formulación ofrece un punto de vista alternativo sobre el aprendizaje, que puede ser más intuitivo o más útil en ciertos escenarios. También es el punto de vista preferido de muchos libros. Damos las ideas básicas en esta

sección, y consideramos aplicaciones concretas más adelante en el libro.

La observación clave es la siguiente. En la sección 3.1, empezamos suponiendo que nuestros ejemplos vienen de una distribución $p(x, y)$. Por la regla del producto de la probabilidad, podemos descomponer $p(x, y)$ como $p(x, y) = p(x) p(y | x)$, de modo que $p(x)$ da la probabilidad de observar cada entrada x , y el término condicional $p(y | x)$ describe la probabilidad de observar cierta salida y dada una entrada x . También podemos descomponerla como $p(x, y) = p(x | y) p(y)$; los métodos que necesitan estimar $p(x)$ o $p(x | y)$ se llaman generativos, mientras que los métodos que estiman $p(y | x)$ se llaman discriminativos. Aparte del modelado del lenguaje, en este libro nos centramos en el segundo caso, y consideramos el modelado generativo de forma más amplia en el próximo volumen. Aproximar $p(y | x)$ con una función $f(x)$ tiene sentido si suponemos que la masa de probabilidad está concentrada sobre todo en torno a un único punto y , es decir, $p(y | x)$ está cerca de una llamada función delta de Dirac, y simplifica drásticamente la formulación general del problema.

SUMA = 1



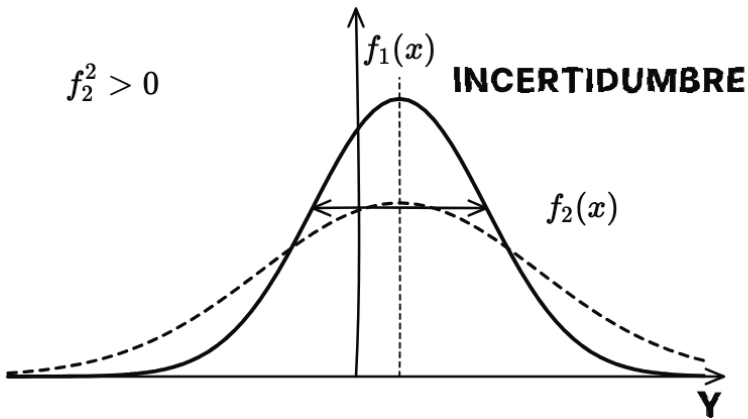
ONE-HOT Y

0
1
0

Un modelo produce una distribución

Sin embargo, podemos relajar esto suponiendo que nuestro modelo $f(x)$ no proporciona la predicción directamente, sino que se usa para parametrizar los estadísticos suficientes de una distribución de probabilidad condicional $p(y \mid f(x))$ sobre las salidas posibles. Por ejemplo, considere un problema de clasificación donde y puede tomar tres valores posibles, 1, 2 o 3. Podemos suponer que nuestro modelo tiene tres salidas que parametrizan una distribución categórica sobre estas clases, de modo que $p(y \mid f(x)) = \prod_{i=1}^3 f_i(x)^{y_i}$. Aquí $y \sim \text{Binary}(3)$ es la codificación one-hot de la clase y , un vector de todos los ceros salvo el elemento de la clase, que es 1, como se introduce formalmente en la sección 4.2, y $f(x) \sim \Delta(3)$ son las probabilidades predichas para cada clase. Como otro ejemplo, suponga que queremos predecir un único valor escalar $y \in \mathbb{R}$, lo que es regresión. Podemos

modelarlo con una función de dos valores $f(x) \sim (2)$ tal que la predicción es una gaussiana con media y varianza apropiadas: $p(y | f(x)) = \mathcal{N}(y | f_1(x), f_2^2(x))$.



La regresión como campana

Aquí la segunda salida de $f(x)$ se eleva al cuadrado para asegurar que la varianza predicha siga siendo positiva. Como puede verse, este es un planteamiento muy general que engloba nuestra discusión anterior, y ofrece más flexibilidad al diseñador, ya que elegir una parametrización concreta para $p(y | x)$ puede ser más fácil que elegir una función de pérdida concreta $l(y, \hat{y})$. Además, este marco proporciona una forma más inmediata de modelar la incertidumbre, como la varianza de la ecuación E.3.6.

Preguntas

1. ¿Por qué una tabla de consulta que memoriza el conjunto de entrenamiento no tiene valor práctico, aunque minimice el riesgo empírico?
 - a. Porque es demasiado lenta
 - b. Porque nuestro verdadero objetivo es una pérdida baja en entradas futuras no vistas, donde solo devuelve un valor por defecto
 - c. Porque las tablas de consulta no pueden derivarse
 - d. Porque necesita un conjunto de prueba
2. ¿Cómo puede estimarse en la práctica la brecha de generalización?
 - a. Calculando exactamente el riesgo esperado
 - b. Comparando la pérdida empírica en el conjunto de entrenamiento con la pérdida en un conjunto de prueba separado nunca usado en el entrenamiento
 - c. Contando los parámetros
 - d. Entrenando dos veces con los mismos datos
3. En la visión probabilística de la clasificación con tres clases, ¿qué produce el modelo?
 - a. Un único número de clase
 - b. Tres probabilidades, una por clase, que parametrizan una distribución categórica
 - c. Un valor de pérdida
 - d. La codificación one-hot de la entrada

4. ¿Por qué se eleva al cuadrado la segunda salida del modelo de regresión?
- a. Para hacer mayor la media
 - b. Para que la varianza predicha de la gaussiana sea siempre positiva
 - c. Para acelerar el entrenamiento
 - d. Para convertir la regresión en clasificación

Respuestas

1. **b)** La memorización alcanza el menor riesgo empírico pero no dice nada sobre entradas nuevas; el conjunto de entrenamiento es solo un sustituto de los datos futuros.
2. **b)** El riesgo esperado no puede calcularse, así que un conjunto de prueba reservado lo aproxima, y la diferencia de pérdidas mide el sobreajuste.
3. **b)** $f(x) \sim \Delta(3)$ da las probabilidades predichas de las tres clases, los estadísticos suficientes de una distribución categórica.
4. **b)** Una varianza debe ser positiva; elevar $f_2(x)$ al cuadrado lo garantiza.

Ejemplos resueltos

Probabilidad de la clase verdadera Un modelo de tres clases predice $f(x) = (0.2, 0.7, 0.1)$. La clase verdadera es la 2. Escriba su codificación one-hot y calcule $p(y | f(x))$ con la fórmula del producto del libro.

La codificación one-hot de la clase 2 tiene un 1 en la posición 2 y ceros en el resto: $y = (0, 1, 0)$.

La fórmula multiplica $f_i(x)$ elevado a y_i para $i = 1, 2, 3$.

Cualquier número elevado a 0 es 1, así que solo cuenta el segundo factor:

$$0.2^0 \times 0.7^1 \times 0.1^0 = 1 \times 0.7 \times 1.$$

El resultado es simplemente la probabilidad que el modelo da a la clase verdadera.

Respuesta: $p(y | f(x)) = 0.7$

Ejercicios

1. Un modelo de regresión produce $f(x) = (3.0, -0.5)$. ¿Qué gaussiana predice para y ?

Pista: La primera salida es la media; la segunda se eleva al cuadrado para obtener la varianza.

Respuesta: Una distribución normal con media 3.0 y varianza $(-0.5)^2 = 0.25$, así que una desviación estándar de 0.5.

2. Un modelo tiene una pérdida media de 0.1 en su conjunto de entrenamiento y de 0.6 en un conjunto de prueba reservado. ¿Cuál es la brecha de generalización aproximada y qué sugiere?

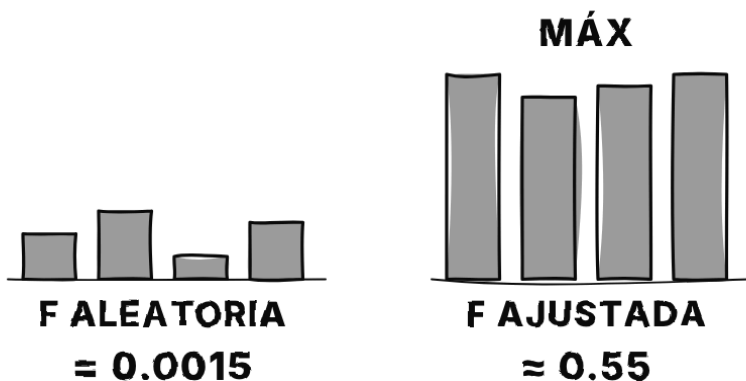
Pista: Reste las dos pérdidas empíricas.

Respuesta: Alrededor de 0.5. El modelo lo hace mucho mejor con los datos que ha visto que con datos nuevos, lo que sugiere que está sobreajustando, tendiendo a la memorización.

Máxima verosimilitud y aprendizaje bayesiano

¿Cómo podemos entrenar un modelo probabilístico? Recuerde que supusimos que las muestras de nuestro conjunto de datos $\mathcal{S}_n$ son muestras i.i.d. de una distribución de probabilidad $p(x, y)$. Por tanto, dado un modelo $f(x)$, la probabilidad que una elección concreta de la función f asigna al propio conjunto de datos viene dada por el producto sobre las muestras del conjunto: $p(\mathcal{S}_n | f) = \prod_{i=1}^n p(y_i | f(x_i))$.

$$p(\mathcal{S}_n | f) = \prod_i p(y_i | f(x_i))$$



Verosimilitud: multiplicar las probabilidades

La cantidad $p(\mathcal{S}_n | f)$ se llama la verosimilitud del conjunto de datos. Para una elección aleatoria de $f(x)$, el modelo asignará probabilidades más o menos al azar entre todas las entradas y salidas posibles, y la verosimilitud de nuestro conjunto de datos concreto será pequeña. Una estrategia sensata, entonces, es seleccionar el modelo de modo que la verosimilitud del conjunto de datos sea, en cambio, máxima. Esta es una aplicación directa del enfoque de máxima verosimilitud, descrito en el apéndice.

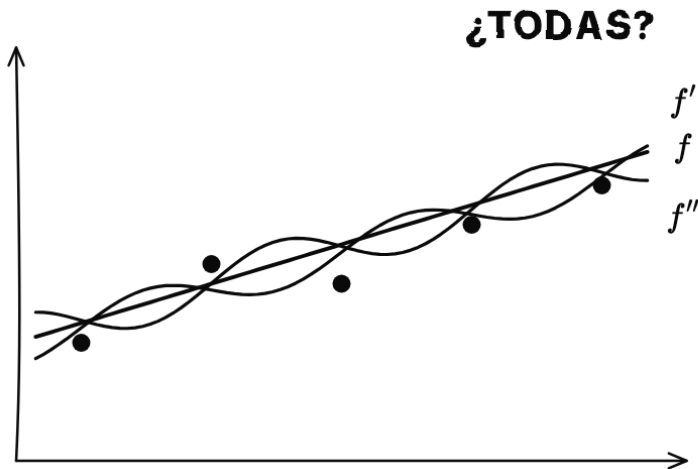
Definición, máxima verosimilitud: dado un conjunto de datos $\mathcal{S}_n$ de pares (x_i, y_i) y una familia de distribuciones de probabilidad $p(y | f(x))$ parametrizadas por $f(x)$, la solución de máxima verosimilitud viene dada por $f^* = \operatorname{argmax}_f \prod_{i=1}^n p(y_i | f(x_i))$.

Aunque de nuevo nos queda un problema de optimización, ahora se sigue directamente de las leyes de la probabilidad una vez elegidas todas las distribuciones de probabilidad, a diferencia de antes, cuando la pérdida concreta formaba parte del espacio de diseño. Los dos puntos de vista, sin embargo, están estrechamente conectados. Trabajando en el espacio logarítmico y pasando a un problema de minimización, obtenemos que el $\operatorname{arg max}$ de $\log \prod_i p(y_i | f(x_i))$ es igual al $\operatorname{arg min}$ de $\sum_i -\log p(y_i | f(x_i))$.

Por tanto, las dos formulaciones son idénticas si identificamos $-\log p(y | f(x))$ como una «pseudopérdida» a optimizar. Como veremos, todas las funciones de pérdida usadas en la práctica pueden obtenerse bajo el principio de máxima verosimilitud para elecciones concretas de este término. Ambos puntos de vista son interesantes, e instamos a los lectores a tenerlos en mente a medida que avanzamos en el libro.

Sección 3.3, Aprendizaje bayesiano. Discutimos aquí una generalización adicional de la formulación probabilística, llamada

redes neuronales bayesianas, BNN, que es de interés en la literatura. Solo damos la idea general, y remitimos al lector a uno de los muchos tutoriales detallados para más información.

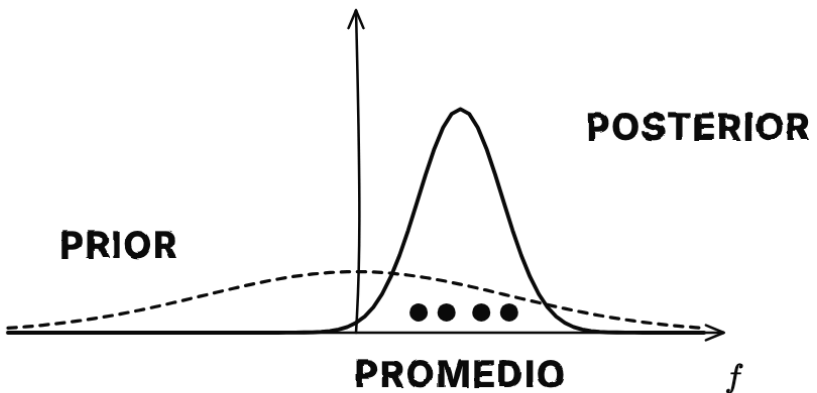


Muchas funciones se ajustan a los datos

Al diseñar una función de probabilidad $p(y | f(x))$ en lugar de $f(x)$ directamente, podemos manejar situaciones donde interesa más de una predicción, es decir, donde la función de probabilidad tiene más de una moda. Sin embargo, nuestro procedimiento sigue devolviendo una sola función $f(x)$ del espacio de todas las funciones posibles, mientras que puede ocurrir que más de una parametrización en todo el espacio de modelos sea válida. En este caso, podría ser útil tener acceso a todas ellas para una predicción más fiel.

Una vez más, podemos lograr este objetivo diseñando otra distribución de probabilidad y dejando luego que las reglas de la probabilidad nos guíen. Como ahora planeamos obtener una distribución sobre todas las funciones posibles, empezamos definiendo una distribución de probabilidad a priori $p(f)$ sobre todas las funciones posibles; recuerde que en la práctica f se describe mediante un conjunto finito de parámetros, en cuyo caso la prior $p(f)$ se convierte en una prior sobre estos pesos. Por ejemplo, veremos que en muchas situaciones se prefieren las funciones de menor norma, por ser más estables, en cuyo caso podríamos definir una prior $p(f) \propto 1/\|f\|$ para alguna norma $\|f\|$ de f .

$$p(f \mid \mathcal{S}_n) = \frac{p(\mathcal{S}_n \mid f) p(f)}{p(\mathcal{S}_n)} \quad = \text{EVIDENCIA}$$



De la prior a la posterior

Una vez observado un conjunto de datos, la probabilidad sobre f se desplaza según la prior y la verosimilitud, y la actualización viene dada por el teorema de Bayes: $p(f \mid \mathcal{S}_n) = p(\mathcal{S}_n \mid f) p(f) / p(\mathcal{S}_n)$, donde $p(f)$ es la prior, antes de observar el conjunto de datos, y $p(f \mid \mathcal{S}_n)$ la posterior, después de observarlo.

El término $p(f \mid \mathcal{S}_n)$ se llama función de distribución posterior, mientras que el término $p(\mathcal{S}_n)$ del denominador se llama la evidencia, y hace falta para asegurar que la posterior esté correctamente normalizada. Suponga por ahora que tenemos acceso a la posterior. A diferencia de antes, la distribución puede codificar una preferencia por más de una sola función f , lo que puede ofrecer mejor poder predictivo. Dada una entrada x , podemos hacer una predicción promediando todos los modelos posibles según su peso posterior: $p(y \mid x) = \int p(y \mid f(x)) p(f \mid \mathcal{S}_n) df$, que es aproximadamente $(1/k) \sum_{i=1}^k p(y \mid f_i(x)) p(f_i \mid \mathcal{S}_n)$.

En la ecuación E.3.10 hemos aproximado la integral con un promedio de Monte Carlo sobre k muestras aleatorias de la distribución posterior, $f_k \sim p(f \mid \mathcal{S}_n)$. La belleza general de este planteamiento se ve empañada por el hecho de que la posterior es en general imposible de calcular en forma cerrada, salvo para elecciones muy concretas de prior y verosimilitud. A falta de esto, uno se ve obligado a aproximar las soluciones, ya sea mediante Monte Carlo con cadenas de Markov o mediante inferencia variacional. Veremos en la sección 9.3.1 un ejemplo de tratamiento bayesiano de los parámetros del modelo, llamado dropout de Monte Carlo.

MAP

$$f^* = \arg \max_f \log p(\mathcal{S}_n | f) + \log p(f)$$

VEROSIM. REGULARIZ.

ML: DATOS AZAR

BAYES: F AZAR

MAP: verosimilitud más regularizador

Señalamos dos hechos interesantes sobre la posterior antes de cerrar esta sección. Primero, suponga que solo nos interesa la función con la mayor densidad posterior. En este caso, el término de evidencia puede ignorarse y la solución se descompone en dos términos separados: $f^* = \operatorname{argmax}_f p(\mathcal{S}_n | f) p(f) = \operatorname{argmax}_f (\log p(\mathcal{S}_n | f) + \log p(f))$, un término de verosimilitud más un término de regularización.

Esto se llama la solución del máximo a posteriori, MAP. Si todas las funciones tienen el mismo peso a priori, es decir, si $p(f)$ es uniforme sobre el espacio de funciones, entonces el segundo término es una constante y el problema se reduce a la solución de máxima verosimilitud. En general, sin embargo, la solución del máximo a posteriori impondrá una penalización a las funciones que se desvíen

demasiado de nuestra distribución a priori. Veremos que esta es una idea útil para combatir el sobreajuste y para imponer restricciones concretas a la función f . El término $\log p(f)$ se llama generalmente un regularizador sobre el espacio de funciones, ya que empuja la solución hacia la cuenca de atracción definida por la distribución a priori. La diferencia entre las soluciones de máxima verosimilitud y de máximo a posteriori está vagamente conectada con la diferencia entre las interpretaciones frecuentista y bayesiana de la probabilidad, es decir, las probabilidades como frecuencias de eventos o las probabilidades como medida de incertidumbre. Desde un punto de vista muy general, la máxima verosimilitud ve los parámetros como un término fijo desconocido y los datos como una muestra aleatoria, mientras que un tratamiento bayesiano ve los datos como fijos y los parámetros como variables aleatorias.

Preguntas

1. ¿Por qué la verosimilitud de un conjunto de datos puede escribirse como un producto sobre sus muestras?
 - a. Porque se supone que las muestras son i.i.d., así que sus probabilidades se multiplican
 - b. Porque todas las muestras tienen la misma etiqueta
 - c. Porque el modelo es lineal
 - d. Porque el conjunto de datos es pequeño
2. ¿Cómo se conecta la máxima verosimilitud con minimizar una pérdida?
 - a. No se conecta; no tienen relación

- b. Tomar logaritmos y cambiar el signo la convierte en minimizar la suma de $-\log p(y_i | f(x_i))$, una pseudopérdida
 - c. Requiere la pérdida de bisagra
 - d. Sustituye al descenso de gradiente
3. ¿Cuándo se reduce la solución del máximo a posteriori a la máxima verosimilitud?
- a. Cuando la prior es uniforme, de modo que $\log p(f)$ es una constante
 - b. Cuando el conjunto de datos está vacío
 - c. Cuando la evidencia es cero
 - d. Nunca
4. ¿Por qué las redes neuronales bayesianas son difíciles de usar de forma exacta?
- a. Porque las priors no pueden definirse
 - b. Porque la posterior es en general imposible de calcular en forma cerrada, así que hacen falta aproximaciones como MCMC o la inferencia variacional
 - c. Porque no tienen verosimilitud
 - d. Porque necesitan conjuntos de prueba etiquetados

Respuestas

1. **a)** Las muestras independientes e idénticamente distribuidas tienen una probabilidad conjunta igual al producto de las probabilidades individuales.
2. **b)** Maximizar el logaritmo del producto es igual a minimizar la suma de los logaritmos negativos de las probabilidades, que hace el papel de una pérdida.

3. a) Con igual peso a priori para todas las funciones, el término de regularización es constante y solo queda la verosimilitud.
4. b) Salvo para priors y verosimilitudes muy concretas, la posterior no tiene forma cerrada.

Ejemplos resueltos

Verosimilitud y logaritmo negativo de la verosimilitud En tres ejemplos, un modelo da a las etiquetas verdaderas las probabilidades 0.9, 0.8 y 0.5. Calcule la verosimilitud del conjunto de datos y el logaritmo negativo de la verosimilitud, usando logaritmos naturales.

Con muestras i.i.d., la verosimilitud es el producto: $0.9 \times 0.8 \times 0.5 = 0.36$.

La pseudopérdida es la suma de $-\log p$ para cada ejemplo: $-\ln 0.9 \approx 0.105$, $-\ln 0.8 \approx 0.223$, $-\ln 0.5 \approx 0.693$.

Su suma es aproximadamente 1.022.

Comprobación: $-\ln 0.36 \approx 1.022$, lo mismo, ya que el logaritmo de un producto es la suma de los logaritmos.

Respuesta: Verosimilitud 0.36; logaritmo negativo de la verosimilitud ≈ 1.022 .

Ejercicios

1. Un segundo modelo da a las mismas tres etiquetas verdaderas las probabilidades 0.6, 0.6 y 0.6. ¿Qué modelo prefiere la máxima verosimilitud?

Pista: Calcule su verosimilitud y compárela con 0.36.

Respuesta: Su verosimilitud es $0.6^3 = 0.216$, menor que 0.36, así que la máxima verosimilitud prefiere el primer modelo.

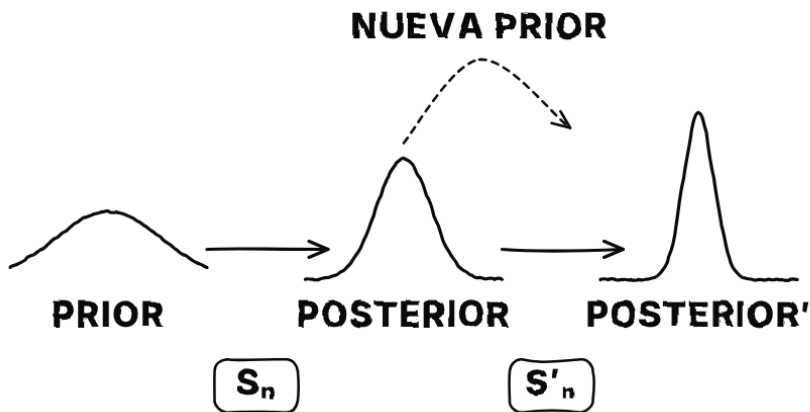
2. Con priors iguales de 0.5 para los dos modelos anteriores, use el teorema de Bayes para hallar la probabilidad posterior del primer modelo.

Pista: Multiplique cada verosimilitud por su prior, luego divida por la suma de ambas, la evidencia.

Respuesta: Prior por verosimilitud: $0.5 \times 0.36 = 0.18$ y $0.5 \times 0.216 = 0.108$. La evidencia es 0.288, así que la posterior del primer modelo es $0.18 / 0.288 = 0.625$.

Modelos lineales y la pérdida cuadrática

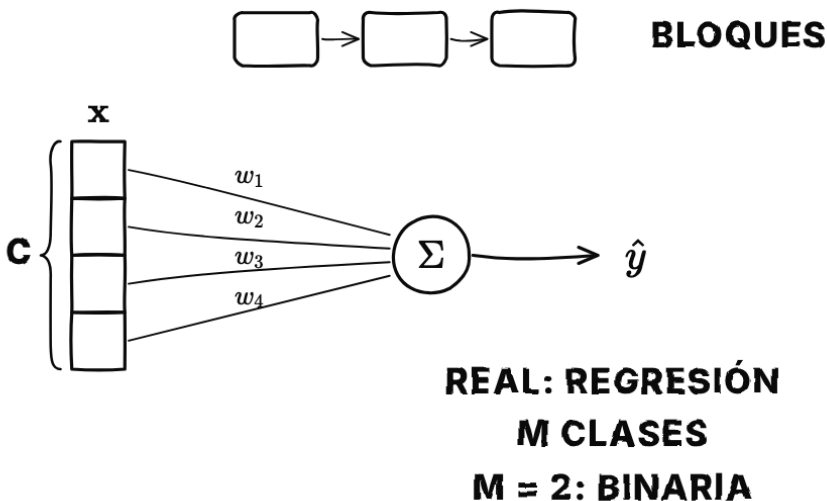
Segundo, el tratamiento bayesiano completo ofrece una forma simple de incorporar datos nuevos, por ejemplo un nuevo conjunto de datos $\mathcal{S}'_n$ de la misma distribución. Para hacerlo, sustituimos la función prior de la ecuación E.3.8 por la distribución posterior que calculamos en la primera porción del conjunto de datos, que ahora representa el supuesto de partida sobre los valores posibles de f , y que se actualiza al mirar los datos nuevos. Piense en la función prior original como la distribución sobre f tras haber observado un conjunto inicial vacío de valores. Esto puede mitigar problemas al entrenar modelos en línea, sobre todo el llamado olvido catastrófico de la información antigua.



SIN OLVIDO

La posterior de ayer, la prior de hoy

Capítulo 4, Modelos lineales. En resumen: programar consiste en elegir la secuencia apropiada de operaciones primitivas para resolver una tarea. Por analogía, construir un modelo consiste en elegir la secuencia correcta de bloques diferenciables. En este capítulo presentamos el bloque más simple, los modelos lineales, que suponen que las entradas actúan aditivamente sobre la salida mediante una media ponderada. En cierto sentido, todos los modelos diferenciables son variaciones y composiciones ingeniosas de bloques lineales.



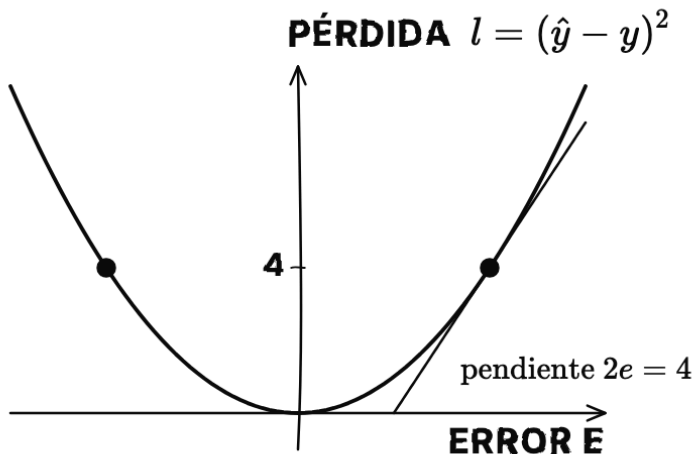
Un modelo lineal: suma ponderada

Sección 4.1, Regresión por mínimos cuadrados. Resumiendo el capítulo anterior, un problema de aprendizaje supervisado puede definirse eligiendo el tipo de entrada x , el tipo de salida y , el modelo f y la función de pérdida l . En este capítulo consideramos las elecciones más simples posibles para todos ellos.

La entrada es un vector $x \sim (c)$, que corresponde a un conjunto de características, por ejemplo c características personales de un cliente de un banco. Usamos el escalar c , abreviatura de canales, para denotar el número de características, por coherencia con los capítulos siguientes. La salida es un único valor real $y \in \mathbb{R}$. En el caso sin restricciones, decimos que es una tarea de regresión. Si y solo puede tomar uno de m valores posibles, de 1 a m , decimos que es una tarea

de clasificación. En el caso especial de $m = 2$, decimos que es una tarea de clasificación binaria. Tomamos f como un modelo lineal, lo que nos da soluciones simples en forma cerrada en algunos casos, sobre todo la regresión por mínimos cuadrados, en la sección 4.1.2.

Las formas básicas que hay que recordar se resumen en la tabla T.4.1: n es el tamaño del conjunto de datos, c el número de características y m el número de clases; por uniformidad, usaremos las mismas letras tanto como sea posible a lo largo del libro. Empezamos discutiendo la elección de la pérdida en el caso de regresión. Partimos del caso de regresión porque, como mostramos más adelante, la clasificación puede resolverse con pequeñas modificaciones del caso de regresión.



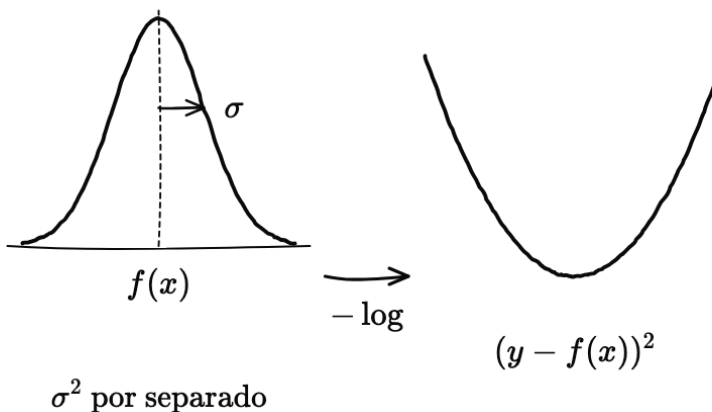
La pérdida cuadrática

La pérdida cuadrática y sus variantes. Encontrar una pérdida para regresión es relativamente simple, ya que el error de predicción $e = \hat{y} - y$ entre la salida predicha del modelo, $\hat{y} = f(x)$, y la salida verdadera deseada y es un objetivo bien definido, al ser una función continua de la salida del modelo que decrece monótonamente. Como en general no nos importa el signo del error de predicción, una elección común es la pérdida cuadrática: $l(\hat{y}, y) = (\hat{y} - y)^2$.

Aquí y en lo que sigue usamos el símbolo $\hat{y}$ para denotar la predicción de un modelo. Como veremos, trabajar con la ecuación E.4.1 aporta varias ventajas a nuestra solución. Entre otras, el gradiente de la pérdida cuadrática es una función lineal de la salida del modelo, lo que nos permite resolver la solución óptima en forma cerrada.

Recordando el principio de máxima verosimilitud, la pérdida cuadrática puede obtenerse suponiendo que las salidas del modelo siguen una distribución gaussiana centrada en $f(\mathbf{x})$ y con una varianza constante σ^2 : $p(y \mid f(\mathbf{x})) = \mathcal{N}(y \mid f(\mathbf{x}), \sigma^2)$.

$$p(y | f(x)) = \mathcal{N}(y | f(x), \sigma^2)$$

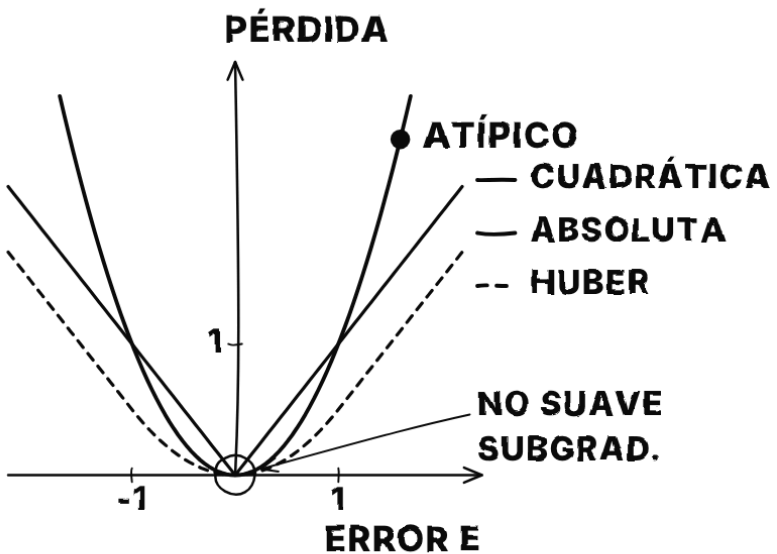


El ruido gaussiano da la pérdida cuadrática

En este caso el logaritmo de la verosimilitud para un solo punto puede escribirse, recordando que el logaritmo de un producto es la suma de los logaritmos y el logaritmo de una potencia es el exponente por el logaritmo, como $\log p(y | f(x), \sigma^2) = -\log \sigma - \frac{1}{2} \log(2\pi) - (y - f(x))^2 / (2\sigma^2)$.

Minimizando la ecuación E.4.2 respecto a f , vemos que los dos primeros términos del lado derecho son constantes, y el tercero vuelve a ser la pérdida cuadrática. Minimizar respecto a σ^2 puede hacerse independientemente de la optimización de f , con una solución simple en forma cerrada, la ecuación E.4.9 más abajo.

Idear variantes de la pérdida cuadrática también es fácil. Por ejemplo, un inconveniente de la pérdida cuadrática es que los errores mayores se penalizan con una fuerza que crece cuadráticamente con el error, lo que puede dar una influencia indebida a los valores atípicos, es decir, puntos muy mal etiquetados. Otras opciones que disminuyen la influencia de los valores atípicos pueden ser la pérdida de valor absoluto, $l(\hat{y}, y) = \text{abs}(\hat{y} - y)$, o la pérdida de Huber, una combinación de la pérdida cuadrática y la pérdida absoluta. La pérdida de Huber es $\frac{1}{2}(y - \hat{y})^2$ cuando el error absoluto es como mucho 1, y $\text{abs}(y - \hat{y}) - \frac{1}{2}$ en otro caso.



Cuadrática, absoluta y Huber

La figura F.4.1 visualiza la pérdida cuadrática, la pérdida absoluta y la pérdida de Huber respecto al error de predicción $e = \hat{y} - y$.

La pérdida de Huber es cuadrática en la proximidad del error 0, y lineal en otro caso, con el término $-1/2$ añadido para asegurar la continuidad.

La pérdida absoluta parece una elección inválida en nuestro contexto, ya que tiene un punto de no diferenciabilidad en 0, debido al valor absoluto. Veremos más adelante que las funciones con uno, o un pequeño número, de puntos de esta forma no son realmente problemáticas. Matemáticamente, pueden manejarse con la noción de subgradiente, una ligera generalización de la derivada. En la práctica, puede imaginar que si partimos de una inicialización aleatoria, el descenso de gradiente nunca alcanzará estos puntos con precisión perfecta, y la derivada del valor absoluto de ϵ para cualquier $\epsilon > 0$ siempre está definida.

Preguntas

1. ¿Bajo qué supuesto la máxima verosimilitud da exactamente la pérdida cuadrática?
 - a. Las salidas siguen una gaussiana centrada en $f(\mathbf{x})$ con varianza constante
 - b. Las salidas siguen una distribución uniforme
 - c. Las salidas son binarias
 - d. El modelo no tiene parámetros
2. ¿Por qué podría preferirse la pérdida de Huber o la absoluta a la pérdida cuadrática?
 - a. Siempre son diferenciables

- b. La pérdida cuadrática crece cuadráticamente con el error, dando a los valores atípicos una influencia indebida; las otras crecen solo linealmente para errores grandes
 - c. Dan soluciones en forma cerrada
 - d. Ignoran el signo del error mientras que la pérdida cuadrática no
3. ¿Por qué la esquina de la pérdida absoluta en 0 no es un problema real en la práctica?
- a. Porque el descenso de gradiente desde un inicio aleatorio prácticamente nunca cae exactamente en 0, y en el resto la derivada existe; los subgradientes manejan el punto matemáticamente
 - b. Porque la pérdida absoluta nunca se usa
 - c. Porque la GPU suaviza la esquina
 - d. Porque la pérdida es constante ahí
4. Según el libro, ¿qué son los modelos lineales para todos los modelos diferenciables?
- a. No tienen relación
 - b. El bloque más simple; todos los modelos diferenciables son variaciones y composiciones ingeniosas de bloques lineales
 - c. Siempre mejores
 - d. Un caso especial solo de la clasificación

Respuestas

1. a) Con una gaussiana de varianza constante, el logaritmo de la verosimilitud tiene dos términos constantes y un tercero proporcional a $(y - f(x))^2$, la pérdida cuadrática.

2. b) La pérdida cuadrática penaliza los errores grandes cuadráticamente, así que los puntos muy mal etiquetados dominan; el crecimiento lineal limita su tirón.

3. a) Un punto de no diferenciabilidad se maneja con subgradientes, y en la práctica nunca se alcanza con precisión perfecta.

4. b) El capítulo abre llamando a los modelos lineales el bloque diferenciable más simple, del que todos los demás son variaciones y composiciones.

Ejemplos resueltos

Tres pérdidas sobre dos errores Calcule las pérdidas cuadrática, absoluta y de Huber para un error pequeño $e = 0.5$ y un error grande $e = 4$.

Pérdida cuadrática: $0.5^2 = 0.25$ y $4^2 = 16$.

Pérdida absoluta: 0.5 y 4 .

Pérdida de Huber: para 0.5 el error absoluto es como mucho 1 , así que es $\frac{1}{2} \times 0.5^2 = 0.125$; para 4 es $4 - \frac{1}{2} = 3.5$.

El error grande pesa 64 veces el pequeño bajo la pérdida cuadrática pero solo 28 veces bajo Huber, por eso Huber limita los valores atípicos.

Respuesta: Cuadrática 0.25 y 16 ; absoluta 0.5 y 4 ; Huber 0.125 y 3.5 .

Ejercicios

1. Compruebe que los dos tramos de la pérdida de Huber se encuentran en un error absoluto de 1 .

Pista: Evalúe ambas fórmulas en el error 1 .

Respuesta: El tramo cuadrático da $\frac{1}{2} \times 1^2 = 0.5$ y el tramo lineal da $1 - \frac{1}{2} = 0.5$: coinciden, que es para lo que está el término $-\frac{1}{2}$.

2. Un modelo bancario usa $c = 12$ características de cliente y un conjunto de datos de $n = 5000$ clientes. ¿Cuál es la forma de una entrada, y predecir un límite de crédito en euros es una tarea de regresión o de clasificación?

Pista: Mire qué tipo de valor toma la salida.

Respuesta: Una entrada es un vector $\mathbf{x} \sim (12)$. Un límite de crédito es un valor real sin restricciones, así que es una tarea de regresión.