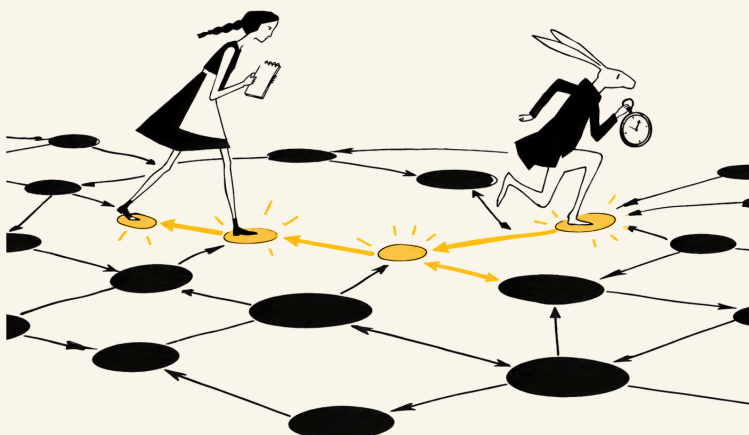


Las aventuras de Alicia en el País de las Maravillas Diferenciables



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

Unas pocas piezas, escaladas sin fin

Ideas viejas, poder nuevo

Cajas de números y sus formas

Flechas que apuntan hacia el mismo lado

Plantillas, lotes y máquinas hechas de números

Cuánto trabajo es una multiplicación

Una receta en letras, y la pendiente de una curva

Pendientes en todas las direcciones

Bajar la montaña en la niebla

Dónde puede terminar el paseo

Una bola con impulso

Del código de práctica a los datos reales

Aprender sin etiquetas, y reutilizar lo aprendido

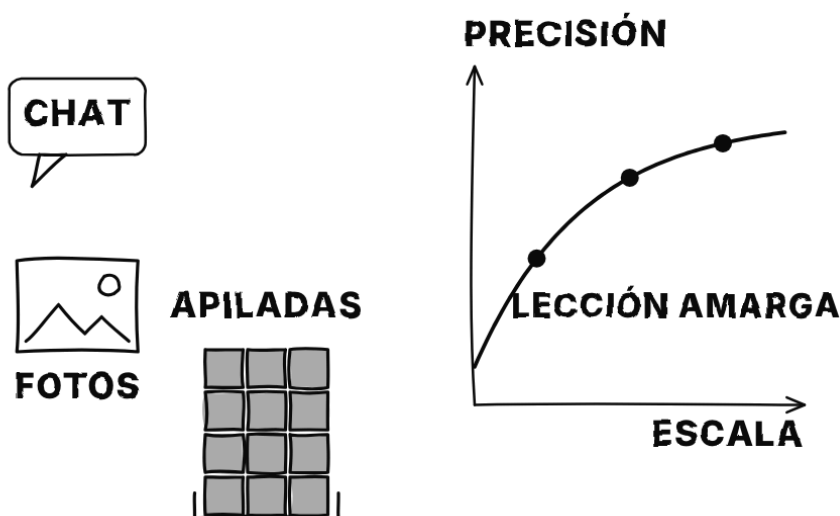
Puntuar una suposición

Memorizar no es aprender

El modelo más creíble

Unas pocas piezas, escaladas sin fin

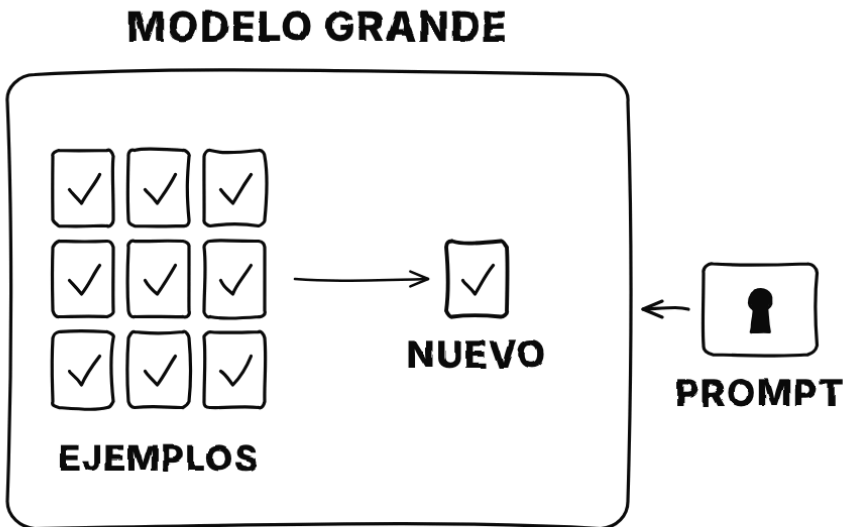
Abra el asistente de chat en su teléfono y pídale que escriba un poema de cumpleaños para su tía; después abra la aplicación de fotos y busque «perro en una playa». Dos tareas muy distintas, una con palabras y otra con imágenes, y las dos resueltas en un segundo. Podría esperar dos máquinas completamente distintas detrás de ellas, cada una construida con su propia receta secreta. La sorpresa de la que parte este libro es la contraria: ambas son redes neuronales, y ambas están construidas con un conjunto muy pequeño de las mismas piezas sencillas. Lo que cambió en los últimos años no son tanto las piezas como la escala a la que se apilan y se entrenan.



Mismas piezas, torres más altas

El libro señala una idea llamada leyes de escalado. A grandes rasgos, para casi cualquier tarea, si le da a un modelo más datos, más potencia de cálculo y más tamaño, todo a la vez, su precisión sube de una forma que se puede predecir de antemano. Dicho al revés, la potencia de cálculo necesaria para alcanzar una precisión dada sigue bajando por un factor constante conforme pasa el tiempo. Por eso las empresas vierten sumas enormes en la inteligencia artificial: la curva ha sido lo bastante fiable para apostar por ella. El investigador Richard Sutton le puso nombre a la lección, la lección amarga: las herramientas simples y generales combinadas con cada vez más potencia de cálculo tienden a ganar a los trucos ingeniosos hechos a mano.

Tomado al pie de la letra, eso suena casi a magia, y el libro tiene cuidado con los puntos donde la magia podría fallar. Una red se entrena para imitar el patrón oculto detrás de los ejemplos que se le muestran, como un falsificador aprende una firma a partir de muchas muestras. Las redes modernas son tan grandes que podrían simplemente memorizar cada ejemplo, como un estudiante que se aprende de memoria la hoja de respuestas y no entiende nada. Sin embargo, en la práctica los modelos entrenados resuelven bien muchas tareas que nunca aparecieron en su entrenamiento. Conforme crecen las colecciones de entrenamiento, la frontera entre preguntas conocidas y desconocidas se difumina, y los modelos grandes parecen apoyarse sorprendentemente poco en la memorización pura.



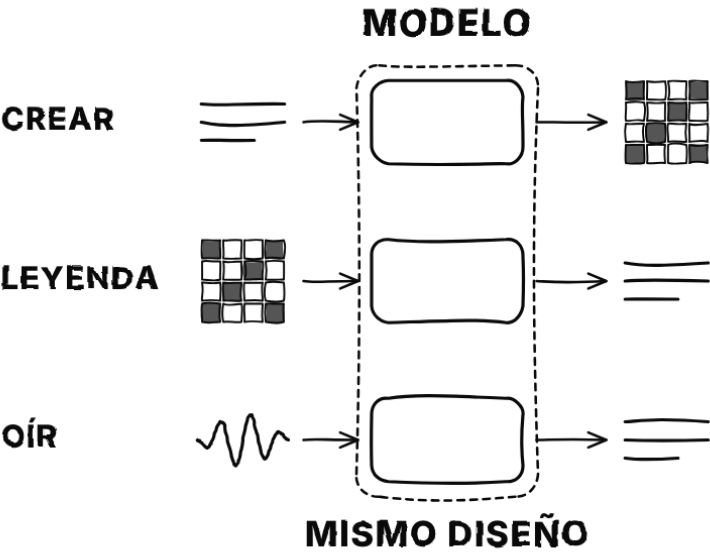
Memorizar o generalizar

Estos modelos tan grandes, que pueden adaptarse a muchas tareas posteriores, se llaman a veces modelos fundacionales, y han cambiado la forma en que la gente usa las redes. Muy a menudo ya no construye nada: escribe una instrucción, un prompt, en un modelo que entrenó otra persona, y su interior sigue siendo una caja cerrada. El libro compara esto con el software corriente. En lugar de escribir cada programa usted mismo, instala uno ya hecho cuyo código fuente nunca ve; y así como solo unas pocas firmas pueden construir enormes programas empresariales, solo unos pocos equipos en el mundo tienen los ordenadores y el saber hacer para entrenar los modelos de lenguaje más grandes, como la familia Llama.

Pero la comparación tiene una segunda mitad, y es la razón de ser del libro. Así como el software de código abierto permite a cualquiera cambiar o reconstruir un programa, el hardware corriente y algo de ingenio permiten a cualquiera experimentar con estos modelos. Puede ajustar uno para su propia tarea, fusionar dos modelos, encoger uno para que funcione en un dispositivo pequeño, probar con qué facilidad se rompe o inventar un diseño nuevo. Para cualquiera de esas cosas necesita mirar bajo el capó: cómo el modelo guarda y transforma los datos por dentro, con todos los trucos que se aprendieron con la experiencia y la depuración. El autor ofrece el libro como ese punto de entrada, y espera que un lector curioso, como Alicia, disfrute del viaje.

El libro supone que ya conoce la idea básica del aprendizaje supervisado: reunir ejemplos del comportamiento que quiere y después entrenar un sistema hasta que los imite. Esa idea tan simple llega sorprendentemente lejos. Hacer imágenes a partir de palabras se convierte en reunir un montón enorme de imágenes con sus pies de foto; imitar el inglés se convierte en aprender a adivinar la siguiente

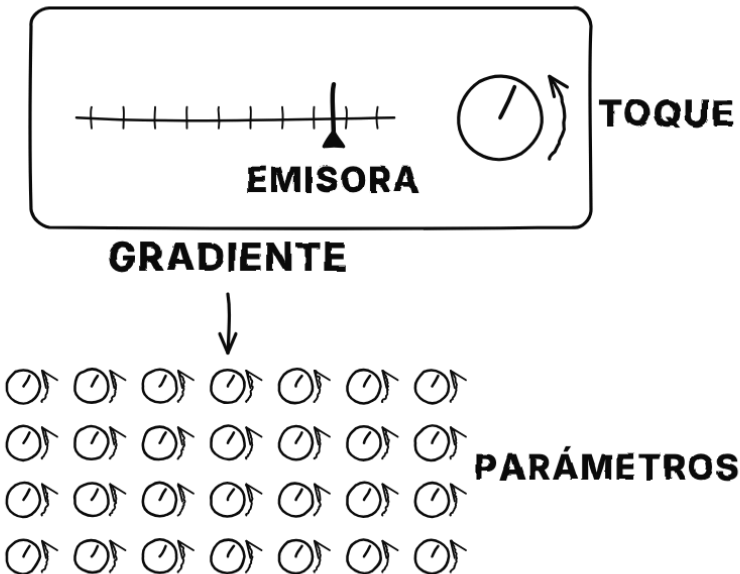
frase a partir de las anteriores, que es como se entrena un asistente de chat; leer una radiografía se convierte en una gran colección de placas con los veredictos de los médicos. La figura del libro ordena las tareas según lo que entra y lo que sale: la generación de imágenes convierte texto en una rejilla de píxeles, el pie de foto va en sentido contrario, y responder a una pregunta hablada convierte sonido en texto. Lo que fascina al autor es que los modelos para todas estas tareas se diseñan siguiendo líneas muy parecidas.



Qué entra, qué sale

Pruébalo

Tome una radio con una sola rueda de sintonía y una emisora desconocida que quiere encontrar. Gire un poco la rueda y escuche: ¿la música suena más clara o más ruidosa que hace un momento? Esa pequeña prueba le dice hacia dónde seguir girando, y tras unos pocos toques está en la emisora sin haber sabido nunca dónde estaba. Ahora imagine no una rueda sino miles de millones, todas movidas a la vez por la misma regla; eso es el entrenamiento.



Mueva la rueda, escuche

El libro llama al aprendizaje una búsqueda. Empieza con un programa que tiene un número enorme de ruedas ajustables, llamadas parámetros, y las va ajustando hasta que el programa se comporta lo bastante bien. Con millones o incluso billones de ruedas, el ensayo y error a ciegas no tiene esperanza, así que la búsqueda necesita una guía. Los modelos diferenciables consiguen esa guía porque están contruidos solo con piezas cuya respuesta a un toque diminuto se puede calcular con exactitud. Saber, para todas las ruedas a la vez, qué haría un giro diminuto es lo que hace posible el ajuste automático, y las dos herramientas que hay detrás, la diferenciación automática y el descenso de gradiente, llenan la primera parte del libro, del capítulo dos al capítulo seis.

Ver una red como una cadena de tales piezas plantea dos preguntas: qué tipos de datos pueden entrar y salir, y qué tipos de piezas podemos usar. El truco del toque necesita cantidades suaves, así que las letras o los números enteros, que saltan de un valor al siguiente, no se pueden usar directamente. Las grandes matrices de números, que el libro llama tensores, son perfectas: una foto es exactamente una matriz así, y se puede remodelar con combinaciones simples de pasos rectos y pasos que curvan. La segunda parte, capítulos siete a nueve, estudia una pieza famosa, la convolución, que trabaja con cualquier cosa dispuesta en orden, como sonido, imágenes, texto y vídeo. Por el camino presenta cómo construir modelos profundos, hechos de muchos pasos uno tras otro, y las ideas de cortar el texto en tokens y generarlo pieza a pieza, la base de los modelos de lenguaje actuales. La tercera parte baja más por la madriguera del conejo: atención y transformadores para conjuntos de elementos en los capítulos diez y once, grafos en el capítulo doce y capas recurrentes para secuencias en el tiempo en el capítulo trece.

Para recordar

El asistente de chat y la aplicación de fotos de su teléfono son ambos redes neuronales construidas con el mismo pequeño conjunto de piezas sencillas. Las leyes de escalado dicen que más datos, más cálculo y modelos más grandes traen de forma predecible mejor precisión, por eso se invierte tanto, y Sutton llamó a esto la lección amarga. Los modelos enormes podrían memorizar sus ejemplos, pero generalizan bien, y la mayoría de la gente los usa ahora mediante prompts como cajas cerradas. Para cambiar o construir tales modelos hay que mirar dentro, y la visión del libro es que el aprendizaje es una búsqueda sobre incontables ruedas, guiada por cómo responde cada pieza a un toque diminuto. El libro pasa después de estas herramientas a las convoluciones para datos ordenados, y a la atención, los grafos y las capas recurrentes.

Preguntas

1. ¿Por qué las empresas invierten sumas tan enormes en hacer los modelos más grandes?
 - a. Porque los modelos más grandes no necesitan ningún dato de entrenamiento
 - b. Porque más datos, cálculo y tamaño juntos han traído de forma predecible mejor precisión
 - c. Porque cada tarea nueva necesita un tipo de red completamente nuevo

- d. Porque los modelos grandes son los únicos que pueden memorizar respuestas
2. Una red enorme podría simplemente memorizar cada ejemplo de entrenamiento. ¿Qué dice el libro que ocurre en realidad?
- a. Los modelos entrenados resuelven bien muchas tareas que nunca aparecieron en su entrenamiento
 - b. Los modelos grandes solo repiten sus ejemplos de entrenamiento palabra por palabra
 - c. Memorizar es imposible, porque las redes son demasiado pequeñas para guardar ejemplos
 - d. Los modelos funcionan solo con preguntas idénticas a sus datos de entrenamiento
3. ¿Por qué insiste el libro en que, para ajustar, fusionar o encoger un modelo, hay que mirar bajo el capó?
- a. Porque usar prompts está prohibido para esos cambios
 - b. Porque solo las empresas más grandes tienen permiso para cambiar modelos
 - c. Porque cambiar un modelo significa entender cómo guarda y transforma los datos por dentro
 - d. Porque los modelos de código abierto no tienen parámetros que ajustar
4. ¿Por qué no se pueden dar letras o números enteros directamente a un modelo diferenciable?
- a. Porque ocupan demasiada memoria
 - b. Porque el truco del toque necesita cantidades suaves, y estas saltan de un valor al siguiente

- c. Porque los tensores solo pueden contener fotos
- d. Porque las convoluciones rechazan todo lo que no sea sonido

Respuestas

1. **b)** Las leyes de escalado dicen que aumentar juntos los datos, la potencia de cálculo y el tamaño del modelo sube la precisión de forma predecible, así que la apuesta es fiable.
2. **a)** Aunque son lo bastante grandes para aprenderse la hoja de respuestas de memoria, los modelos entrenados generalizan bien y se apoyan sorprendentemente poco en la memorización pura.
3. **c)** Usar un modelo como caja cerrada basta para escribir prompts, pero personalizar o construir uno exige saber cómo procesa los datos por dentro, con sus trucos aprendidos.
4. **b)** El entrenamiento se apoya en saber qué hace un toque diminuto, lo que exige cantidades suaves; las letras y los números enteros son discretos, mientras que las grandes matrices de números, como las fotos, funcionan bien.

Ejercicios

1. Explique con sus propias palabras por qué el libro llama al aprendizaje una búsqueda, y por qué esa búsqueda necesita piezas diferenciables cuando un modelo tiene miles de millones de parámetros.

Pista: Piense en la rueda de la radio: ¿qué le dice un giro pequeño, y qué pasa cuando hay miles de millones de ruedas?

Respuesta: Un modelo es un programa con un número enorme de ruedas ajustables, y aprender significa ajustarlas hasta que el programa se comporta bien. Con miles de millones de ruedas, el ensayo y error al azar nunca terminaría. Si cada pieza responde a un toque diminuto de una

forma que se puede calcular con exactitud, sabemos para todas las ruedas a la vez hacia dónde girar, y herramientas automáticas como el descenso de gradiente pueden hacer el ajuste.

2. El libro dice que muchas tareas muy distintas se convierten en el mismo tipo de problema bajo el aprendizaje supervisado. Tome la creación de imágenes a partir de palabras y explique qué son los «ejemplos» y qué aprende el modelo.

Pista: Pregúntese qué entra, qué debería salir y qué colección de pares reuniría.

Respuesta: La entrada es un texto y la salida es una imagen, así que los ejemplos son un montón muy grande de imágenes con sus pies de foto. El modelo se entrena hasta que, dado un pie de foto, produce una imagen que encaja, imitando el comportamiento mostrado en los pares reunidos.

3. Usando la comparación del libro con el software, explique los dos lados: por qué la mayoría de la gente usa ahora los modelos grandes como cajas cerradas, y por qué la gente corriente aún puede experimentar con ellos.

Pista: Un lado trata de quién puede construir los modelos más grandes; el otro, del código abierto.

Respuesta: Como el gran software empresarial, los modelos más grandes solo pueden construirlos unos pocos equipos con ordenadores enormes y saber hacer, así que la mayoría de la gente simplemente escribe prompts a un modelo que entrenó otra persona. Pero así como cualquiera puede cambiar o reconstruir el software de código abierto, el hardware corriente y algo de ingenio permiten ajustar, fusionar, encoger o probar modelos, siempre que se entienda lo que pasa dentro.

Ideas viejas, poder nuevo

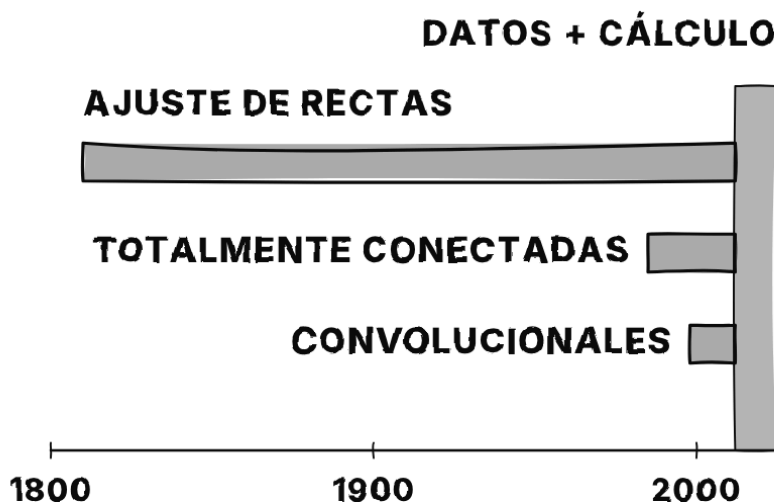
Imagine una foto de periódico de 1958: un hombre serio con camisa blanca junto a un armario lleno de cables, bajo un titular que promete una máquina que aprenderá como un cerebro. El hombre es el psicólogo estadounidense Frank Rosenblatt, y la máquina, el perceptrón, fue una de las primeras redes neuronales que funcionaron. El libro muestra esta escena a propósito, con un guiño: bombo de la inteligencia artificial, solo que en 1958. La emoción que ve hoy alrededor de los asistentes de chat no es la primera ola, y las ideas que hay debajo son más viejas de lo que la mayoría piensa.

El propio nombre «red neuronal» es un resto de los primeros intentos del siglo veinte de imitar las neuronas de los cerebros vivos. Palabras de aquella época se quedaron con nosotros, como neuronas, capas y activaciones, y el libro las conserva para coincidir con el software que todos usan. El interés llegó en olas. Entre 2012 y 2017, grandes desafíos públicos, sobre todo la competición ImageNet de reconocer objetos en fotos, empujaron a las redes a crecer deprisa en tamaño y complejidad. Una segunda ola empezó en 2017 con los transformadores, que se hicieron con el campo del lenguaje en muy poco tiempo, igual que una familia de modelos se había hecho con la visión por ordenador unos años antes. Siguieron avances en vídeo, grafos y sonido, que llevaron a la emoción actual por los grandes modelos de lenguaje, los modelos que mezclan texto e imágenes y los modelos que generan.

El vocabulario cambió con cada ola, del conexionismo de los años ochenta al aprendizaje profundo, que distinguió las redes modernas

de los modelos más pequeños y menos profundos del pasado. El autor encuentra vagas todas estas etiquetas, porque las redes artificiales modernas casi no tienen nada que ver con los cerebros reales. Lo que de verdad tienen en común es que están construidas con bloques que se pueden diferenciar, bloques cuya respuesta a un toque diminuto se puede calcular. Por eso el libro prefiere el nombre más llano de modelos diferenciables. Vistas así, las redes neuronales pertenecen a una disciplina más amplia y joven llamada programación diferenciable, que mezcla informática y optimización para estudiar cualquier programa de ordenador que pueda afinarse de este modo.

Recorrer esta tierra es también recorrer la historia. Ajustar modelos simples de línea recta mediante pequeñas correcciones repetidas, el método del cuarto capítulo del libro, se conocía al menos desde el siglo diecinueve. Las redes totalmente conectadas que se usan más adelante en el libro vienen de los años ochenta, y los modelos convolucionales para imágenes ya se usaban a finales de los noventa. Lo que llevó décadas no fueron las ideas, sino reunir suficientes datos y suficiente potencia de cálculo para mostrar lo bien que funcionan estos modelos cuando son lo bastante grandes. El teléfono que reconoce su cara funciona con ideas más viejas de lo que podría suponer, hechas prácticas solo hace poco.



Ideas viejas que esperan poder

El libro no puede cubrirlo todo, en parte porque la investigación avanza muy deprisa, pero pretende darle lo suficiente para orientarse por su cuenta en los artículos más nuevos. Más allá de los capítulos impresos, el autor promete una página web con capítulos extra sobre temas que no dependen de un solo tipo de datos, como los modelos generativos, los modelos que encienden y apagan partes de sí mismos, reutilizar lo que un modelo aprendió para otra tarea y explicar lo que hacen los modelos. Esos capítulos se inclinan hacia la investigación y se pueden leer en cualquier orden. También hay sesiones prácticas guiadas en forma de cuadernos, que siguen gran parte del libro y van más allá, por ejemplo al aprendizaje comparando pares de ejemplos y a la fusión de modelos.

Para recordar

El bombo alrededor de las máquinas que aprenden no es nuevo: en 1958 el perceptrón de Rosenblatt ya salía en los titulares. Nombres como neurona y capa son restos históricos, y las olas de interés llegaron con ImageNet desde 2012 y con los transformadores desde 2017. Como las redes modernas se parecen muy poco a los cerebros, el libro las llama modelos diferenciables, parte del campo más amplio de la programación diferenciable. La mayoría de las ideas clave tienen décadas; lo que faltaba eran suficientes datos y potencia de cálculo para mostrar su fuerza.

Preguntas

1. ¿Por qué el libro prefiere el nombre «modelos diferenciables» al de «redes neuronales»?
 - a. Porque las redes modernas copian las neuronas reales con más fidelidad que antes
 - b. Porque lo que de verdad comparten las redes modernas es estar construidas con bloques que se pueden diferenciar
 - c. Porque la palabra «neuronal» está prohibida en el software actual
 - d. Porque los modelos diferenciables son un invento completamente nuevo de 2017

2. Si las ideas clave tienen décadas, ¿qué frenó a estos modelos durante tanto tiempo?
 - a. Nadie había inventado aún la idea de corregir un modelo paso a paso
 - b. Los modelos fueron prohibidos tras el bombo de 1958
 - c. No había suficientes datos ni potencia de cálculo para mostrar lo bien que funcionan los modelos grandes
 - d. Las redes totalmente conectadas se inventaron solo en 2012
3. ¿Por qué el libro sigue usando palabras como neuronas, capas y activaciones?
 - a. Para coincidir con los términos del software existente, aunque sean restos históricos
 - b. Porque las redes modernas son modelos exactos del cerebro
 - c. Porque la programación diferenciable no tiene palabras propias
 - d. Porque la competición ImageNet exigía esos nombres

Respuestas

1. **b)** Las redes artificiales modernas casi no tienen nada que ver con los cerebros; su verdadero rasgo común es que están construidas con bloques diferenciables.
2. **c)** El ajuste de líneas rectas data del siglo diecinueve, las redes totalmente conectadas de los años ochenta y los modelos convolucionales de los noventa; lo que faltaba eran datos y potencia de cálculo.
3. **a)** Los nombres vienen de los primeros intentos de imitar las neuronas del cerebro y siguieron en uso, así que el libro los conserva para ser coherente con las bibliotecas que todos usan.

Ejercicios

1. Explique con sus propias palabras qué quiere decir el libro con la broma «bombo de la IA, solo que en 1958».

Pista: Piense en quién era Rosenblatt y qué era el perceptrón.

Respuesta: En 1958 el perceptrón de Frank Rosenblatt, una de las primeras redes neuronales que funcionaron, atrajo mucha atención de los medios, de forma muy parecida a la emoción actual por la IA. La broma nos recuerda que el entusiasmo por las máquinas que aprenden llega en olas y que las ideas básicas son mucho más viejas que el auge actual.

2. Describa las dos grandes olas de interés por las redes neuronales que el libro nombra a partir de 2012, y qué cambió cada una.

Pista: Una ola la impulsó una competición sobre fotos; la otra, un nuevo tipo de modelo para el lenguaje.

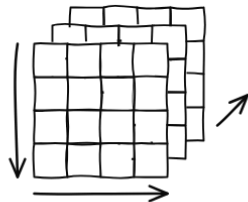
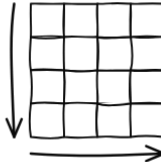
Respuesta: De 2012 a 2017, grandes desafíos como ImageNet empujaron a las redes de reconocimiento de imágenes a crecer deprisa en tamaño y complejidad. Desde 2017, los transformadores se hicieron rápidamente con el procesamiento del lenguaje, igual que antes los modelos convolucionales se habían hecho con la visión, y nuevos avances en vídeo, grafos y sonido llevaron a la emoción actual por los grandes modelos de lenguaje y los modelos generativos.

Cajas de números y sus formas

Tome una hoja de papel cuadriculado y escriba un solo número en una casilla. Después escriba una columna de números hacia abajo, luego rellene todo un rectángulo, y por último apile varias hojas rellenas una sobre otra. Acaba de construir los cuatro tipos de objetos de los que está hecho todo el libro. Un solo número, una columna, una rejilla y una pila de rejillas se diferencian en una sola cosa: cuántas direcciones necesita para encontrar una casilla.

ESCALAR VECTOR MATRIZ

7

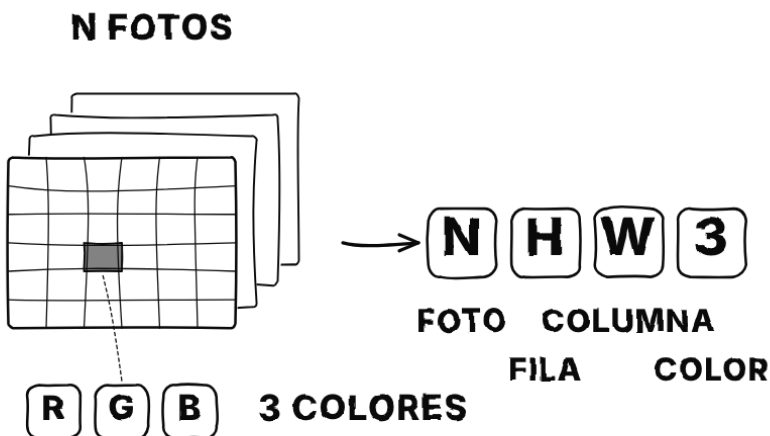


RANGO 0 RANGO 1 RANGO 2 RANGO 3

Tensores de rango 0 a 3

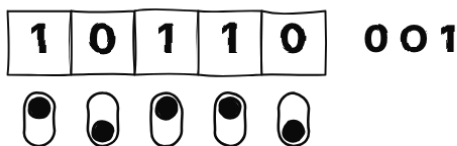
El libro los llama a todos tensores, y al número de direcciones lo llama el rango del tensor. Un número solo tiene rango cero y se llama escalar; una columna de números tiene rango uno y se llama vector; un rectángulo de números tiene rango dos y se llama matriz; lo que tiene más direcciones no tiene nombre especial. El autor admite que la palabra tensor significa algo más estricto para los matemáticos, y les pide perdón, pero el sentido laxo es tan común en el software que el libro lo conserva. Cada tipo recibe en el libro su propio estilo de letra, para que de un vistazo sepa cuál está mirando, y las letras caligráficas se reservan para las colecciones y las probabilidades.

Ahora piense en las fotos de su teléfono, cargadas en un modelo de una sola vez. Cada foto es una rejilla de píxeles, cada píxel guarda tres números para el rojo, el verde y el azul, y hay muchas fotos a la vez. Así que todo el lote es una caja con cuatro direcciones: qué foto, qué fila, qué columna y cuál de los tres colores. El libro escribe una caja así listando sus tamaños, la forma, y deja algunos como letras, porque el número de fotos o su altura pueden cambiar mientras que los tres colores nunca cambian. El consejo clave del autor es que entender una operación muy a menudo se reduce a entender la forma de cada caja que entra y que sale.

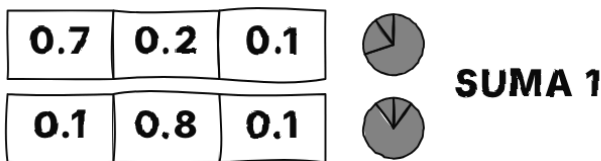


Un lote de fotos como caja

El mismo signo corto que anuncia una forma se usa también, rara vez, para sacar un número aleatorio de una distribución de probabilidad; el autor promete que el significado siempre quedará claro por el contexto. Dos etiquetas más le dicen qué puede haber dentro de una caja. Una dice que la caja contiene solo ceros y unos, como una fila de interruptores de luz que están apagados o encendidos. La otra dice que los números nunca son negativos y suman uno, como las porciones de un gráfico circular, o como la confianza que una aplicación de fotos reparte entre las etiquetas gato, perro y coche. Para cajas más grandes de ese segundo tipo, cada fila por sí sola suma uno, así que cada foto tiene su propio gráfico circular.



GATO PERROCOCHE



$$1 \text{ FOTO} = H \times W \times 3$$

Interruptores y gráficos

Pruébalo

Abra el panel de información de cualquier foto de su teléfono y busque su tamaño, por ejemplo un ancho y un alto en píxeles. Multiplíquelos mentalmente y añada que cada píxel guarda tres valores de color: ese es el tamaño de la caja que el modelo ve para una foto. Ahora imagine cien fotos así enviadas juntas, y tiene una caja con cuatro direcciones, exactamente la clase que describe el libro.

El libro nació de un curso sobre redes neuronales para ciencia de datos que el autor lleva muchos años impartiendo en la Universidad

Sapienza de Roma, con material añadido de otros cursos, tutoriales y escuelas de verano. Empieza en el mismo terreno que otros buenos libros recientes, y después sigue los intereses de investigación del propio autor, ofreciendo un punto de vista complementario. Su convicción central es que estos modelos se entienden a través de la teoría y el código a la vez, y la diferenciación automática es el mejor ejemplo de ese constante ir y venir. El libro atribuye buena parte del regreso de las redes neuronales desde 2012 a potentes bibliotecas de software, desde las primeras como Theano, Caffe y Chainer hasta las actuales TensorFlow, PyTorch y JAX. Conecta sus ideas con PyTorch y JAX siempre que puede, pero no es un manual de programación, y para eso le remite a la documentación de cada biblioteca.

El autor también es honesto sobre lo que el libro no es. Elige ideas que son comunes hoy y lo bastante generales para durar, pero no puede ver el futuro, y algunas partes pueden estar desfasadas cuando usted las lea. Para cada idea muestra unas pocas variantes que aparecen en artículos de investigación, como dos formas distintas de mantener bien escalados los números en una red, pero existen cientos más, y una gran página web pública que recopila artículos junto con su código es el lugar para explorarlas. Por último, hacer que estos modelos funcionen a gran escala exige habilidad de ingeniería y algo de intuición: el libro dice poco sobre hardware, y para la intuición nada supera a la experiencia y a las entradas de blog francas de quienes lo practican.

Para recordar

Todo en el libro es un tensor, una caja de números cuyo rango es el número de direcciones que necesita para encontrar una entrada: un número, una columna, una rejilla, o más. Un lote de fotos en color es una caja con cuatro direcciones, y entender una operación suele significar conocer la forma de cada caja implicada. Algunas cajas contienen solo ceros y unos; otras, números no negativos que suman uno, como un gráfico circular de confianza. El libro viene de un curso universitario de muchos años, una teoría y código, se apoya en PyTorch y JAX, y admite que la ingeniería y la experiencia deben completar el resto.

Preguntas

1. ¿Qué hace que un escalar, un vector y una matriz sean tipos distintos de tensor?
 - a. El tamaño de los números que contienen
 - b. Cuántas direcciones necesita para encontrar una entrada
 - c. Si los números son enteros o tienen decimales
 - d. Si vienen de fotos o de texto
2. Un modelo recibe un lote de fotos en color de una sola vez. ¿Por qué ese lote es un tensor con cuatro direcciones?
 - a. Porque cada foto tiene cuatro esquinas
 - b. Porque el modelo siempre necesita exactamente cuatro fotos

- c. Porque hay que decir qué foto, qué fila, qué columna y cuál de los tres colores
 - d. Porque las fotos en color se guardan como cuatro archivos separados
3. Una aplicación de fotos informa de lo segura que está de que una imagen muestra un gato, un perro o un coche. ¿Qué etiqueta del libro encaja con esos números?
- a. Números que nunca son negativos y suman uno, como porciones de un gráfico circular
 - b. Una caja que contiene solo ceros y unos
 - c. Un tensor de rango tres sin restricciones
 - d. Un solo escalar sin forma
4. Según el autor, ¿qué es lo que más a menudo desbloquea la comprensión de una operación en un modelo?
- a. Memorizar los nombres de todas las bibliotecas de software
 - b. Conocer la forma de cada tensor que entra y que sale
 - c. Leer las especificaciones del hardware
 - d. Contar el número de notas al pie del capítulo

Respuestas

- 1. b)** Un número, una columna, una rejilla y una pila de rejillas se diferencian solo en cuántas direcciones necesita para localizar una casilla, y ese número es el rango.
- 2. c)** Cada foto es una rejilla de píxeles con tres valores de color por píxel, y llegan muchas fotos juntas, así que encontrar un número exige cuatro elecciones.

3. a) Las confianzas repartidas entre etiquetas son no negativas y suman uno, como las porciones de un gráfico circular; para muchas fotos, cada fila tiene su propio gráfico.

4. b) El consejo clave del libro es que entender una operación suele reducirse a entender la forma de cada tensor implicado.

Ejercicios

1. Un modelo lee un lote de clips de sonido cortos. Cada clip es una larga fila de valores de volumen, y todos los clips del lote tienen la misma longitud. ¿Cuántas direcciones tiene el lote y qué representa cada dirección?

Pista: Pregúntese qué debe elegir para señalar un único valor de volumen.

Respuesta: Dos direcciones: qué clip y qué instante dentro del clip.

Necesita ambas elecciones para encontrar un valor, así que el lote es una rejilla de números, un tensor de rango dos, una matriz.

2. Explique por qué el libro deja algunos tamaños de una forma como letras pero escribe otros como números fijos, usando como ejemplo el lote de fotos en color.

Pista: ¿Qué tamaños pueden cambiar de un lote al siguiente, y cuáles no pueden cambiar nunca?

Respuesta: El número de fotos y su altura y anchura pueden variar de un uso a otro, así que se dejan como letras; cada píxel tiene siempre exactamente tres valores de color, así que ese tamaño se escribe como un número fijo.

3. Diga con sus propias palabras qué admite el libro que no es, y adónde envía al lector para esas cosas.

Pista: Piense en los detalles de programación, la completitud y en hacer que los modelos funcionen a gran escala.

Respuesta: No es un manual de programación, así que remite a la documentación de PyTorch y JAX. No es completo ni a prueba de futuro, así que para las muchas variantes de cada idea remite a una página web pública que recopila artículos con su código. Y dice poco sobre hardware y sobre el oficio de hacer que los modelos funcionen a gran escala, para lo cual la experiencia y las entradas de blog de quienes lo practican son los mejores maestros.

Flechas que apuntan hacia el mismo lado

Sostenga dos lápices delante de usted, los dos partiendo de la misma punta del dedo. Apúntelos exactamente en la misma dirección, después ábralos en ángulo recto, después gire uno hasta que apunten en direcciones opuestas. Sin ningún número, ya puede decir cuánto están de acuerdo los dos lápices: del todo, nada, o completamente enfrentados. Al final de esta parte verá que una operación simple mide exactamente ese acuerdo, y que está en el corazón de casi todos los modelos del libro.



ACUERDO



NADA



OPOSICIÓN

Acuerdo, nada, oposición

Primero, unas palabras sobre el propio libro, tomadas de sus agradecimientos. Las ecuaciones en color vienen de un paquete de composición gratuito de otro autor, las imágenes en color de Alicia y los pequeños símbolos del margen tienen licencia de una agencia de fotos de archivo, y los dibujos de Alicia dentro de los capítulos son las ilustraciones originales de John Tenniel, tomadas de Wikimedia. El autor agradece a los colegas y lectores que enviaron correcciones y le animaron a publicar.

La licencia importa para lo que está escuchando ahora. El libro se publica bajo una licencia Creative Commons que permite a cualquiera compartirlo, remezclarlo, adaptarlo y construir sobre él, en cualquier forma e incluso comercialmente, siempre que se dé crédito al autor. Hay una condición: lo que se construya sobre él debe publicarse bajo los mismos términos. Este curso es exactamente una remezcla así, de modo que lleva la misma licencia, y no reutiliza las imágenes de archivo con licencia, que la licencia del libro no cubre.

El capítulo dos reúne después las matemáticas que necesita el resto del libro. El autor supone que ya se ha encontrado antes con estos temas, y usa el capítulo sobre todo para acordar la notación y dar una visión de conjunto conectada, señalando dónde una idea se encuentra con el código real. Avanza en tres pasos, cada uno apoyado en el anterior: álgebra lineal, después gradientes para cajas de números de cualquier rango, después cómo se usan los gradientes para mejorar una función. La probabilidad tiene su propio apéndice, construido alrededor del principio de máxima verosimilitud, y el autor avisa, con amabilidad, de que el capítulo está denso de definiciones.

Formalmente, un tensor es una caja de elementos que tienen todos el mismo tipo, y se describe por ese tipo y por su forma, la lista de sus

tamaños que vimos antes. Los tensores importan tanto porque se adaptan al hardware masivamente paralelo: los chips gráficos de los ordenadores para juegos, y más aún los chips especializados, pueden trabajar sobre miles de entradas al mismo tiempo. Cuando un asistente de chat le responde en segundos, está empujando enormes cajas de números a través de esos chips. La mayoría de las cajas del libro contienen números con decimales, pero pueden contener números enteros, por ejemplo etiquetas de clase, o incluso trozos de texto. Puede cortar rebanadas de una caja, una fila, un bloque, una sola entrada, con las convenciones de la popular biblioteca NumPy, salvo que el libro cuenta desde uno en lugar de desde cero.

	+	10	20	30	40
1		11	21	31	41
2		12	22	32	42
3		13	23	33	43
4		14	24	34	44

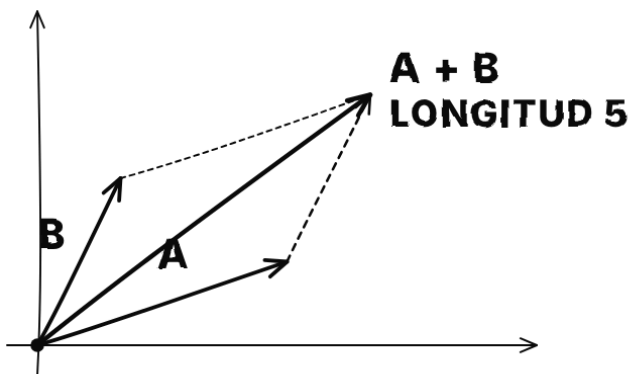
16, NO 4

Cuatro más cuatro son dieciséis

Casi todos los modelos del libro son una cadena de operaciones muy corrientes: sumas, multiplicaciones y unas pocas funciones que curvan, como la exponencial, el seno, el coseno y la raíz cuadrada. Los libros de texto distinguen con cuidado entre un vector escrito como columna y el mismo vector escrito como fila. En el código la distinción es más delicada, porque una columna, una fila y una simple lista de números son tres formas distintas. Las bibliotecas siguen una regla llamada *broadcasting*: cuando dos cajas no encajan, la más pequeña se repite en silencio hasta que encajan. El libro muestra un accidente clásico: suma una columna de cuatro números a una simple lista de cuatro números, espera cuatro resultados, y obtiene en su lugar una rejilla de dieciséis, sin ningún mensaje de error.

Pruébalo

Dibuje una pequeña cruz en una hoja de papel para el origen, y dibuje dos flechas que salgan de ella. Ahora deslice una copia de la segunda flecha para que empiece en la punta de la primera, y dibuje la flecha desde el origen hasta donde ha terminado. Acaba de sumar dos vectores: el resultado es la diagonal del paralelogramo que forman las dos flechas. Mida el resultado con una regla: esa longitud en línea recta desde el origen es lo que el libro llama la norma.

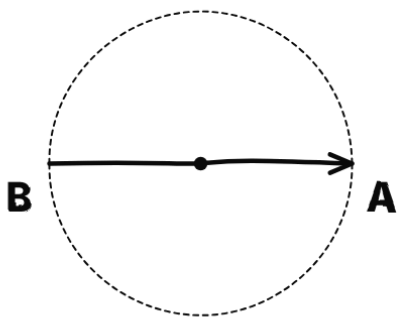


Sumar flechas

Esa es la imagen geométrica detrás de los vectores. Dos vectores cualesquiera de la misma forma se pueden estirar y sumar para dar un tercero, entrada por entrada, y esa libertad es lo que los matemáticos llaman un espacio vectorial. Visto como un punto en el espacio, un vector tiene una longitud, su distancia al origen, y el libro usa a menudo la longitud al cuadrado, que es simplemente todas las entradas elevadas al cuadrado y sumadas.

La operación vectorial más importante de todo el libro es el producto escalar, también llamado producto interno. Multiplica los dos vectores entrada por entrada y suma todo lo obtenido, así que el resultado es un solo número, no un vector. El pequeño ejemplo del libro multiplica dos listas cortas de tres números y termina con un

número ligeramente negativo. Su significado es geométrico: el producto escalar es las dos longitudes multiplicadas entre sí, por una medida de lo bien alineadas que están las dos flechas. Cuando ambas flechas tienen longitud uno, lo único que queda es la alineación misma, llamada similitud coseno. Es máxima cuando las flechas apuntan hacia el mismo lado, mínima cuando apuntan en direcciones opuestas, y exactamente cero cuando forman un ángulo recto, lo que el libro llama ortogonales. Las funciones de búsqueda que encuentran fotos o mensajes parecidos en significado a su consulta suelen convertir cada elemento en una flecha así y comparar las direcciones de este modo.



$$\mathbf{A} \cdot \mathbf{B} = \sum \mathbf{A}_i \mathbf{B}_i$$

$$\text{COS} = -1.00$$

Similitud coseno

Para recordar

Los agradecimientos del libro dan crédito al paquete de ecuaciones, a las imágenes de archivo con licencia y a los dibujos originales de Alicia de Tenniel. Su licencia permite a cualquiera remezclarlo, incluso comercialmente, con crédito y bajo los mismos términos, por eso existe este curso y por eso evita las imágenes de archivo. El capítulo dos repasa álgebra lineal, gradientes y optimización; los tensores son cajas de un solo tipo con una forma, ideales para chips paralelos, y el broadcasting puede convertir en silencio cuatro resultados en dieciséis. El producto escalar multiplica dos vectores entrada por entrada y suma, dando un número que mide a la vez las longitudes y la alineación. Para flechas de longitud uno es la similitud coseno: máxima para la misma dirección, mínima para direcciones opuestas, cero para un ángulo recto.

Preguntas

1. Dos flechas de longitud uno forman un ángulo recto entre sí. ¿Cuál es su similitud coseno?
 - a. Su valor más alto, porque ambas flechas tienen longitud uno
 - b. Su valor más bajo, porque apuntan en direcciones distintas
 - c. Exactamente cero, porque son ortogonales
 - d. No se puede decir sin conocer cada entrada

2. Suma una columna de cuatro números a una simple lista de cuatro números y obtiene una rejilla de dieciséis números. ¿Qué ha pasado?
 - a. La biblioteca cometió un error aritmético
 - b. El broadcasting repitió en silencio las cajas para que sus formas encajaran
 - c. Se calculó el producto escalar en lugar de la suma
 - d. Los números eran demasiado grandes y se desbordaron
3. ¿Por qué son tan centrales los tensores en el aprendizaje profundo?
 - a. Porque se adaptan a chips masivamente paralelos que trabajan sobre miles de entradas a la vez
 - b. Porque solo pueden contener números con decimales
 - c. Porque convierten cada vector en una columna
 - d. Porque eliminan la necesidad de gradientes
4. ¿Qué mide el producto escalar de dos vectores, en la lectura geométrica del libro?
 - a. Solo la diferencia entre sus longitudes
 - b. El número de entradas que tienen en común
 - c. Sus dos longitudes multiplicadas entre sí, por lo bien alineadas que están las flechas
 - d. El área del paralelogramo que forman

Respuestas

1. c) La similitud coseno es máxima para la misma dirección, mínima para direcciones opuestas y exactamente cero para un ángulo recto, lo que el libro llama ortogonales.

2. b) Una columna y una simple lista son formas distintas, y el broadcasting repite en silencio la caja más pequeña hasta que las formas encajan, dando una rejilla en lugar de cuatro resultados.

3. a) Las cajas de números de un solo tipo son ideales para los chips gráficos y otro hardware especializado que procesa muchas entradas al mismo tiempo.

4. c) El producto escalar es igual al producto de las longitudes por una medida de alineación; para flechas de longitud uno solo queda la alineación, la similitud coseno.

Ejemplos resueltos

El producto escalar del libro El libro toma dos vectores cortos, el primero con entradas 0,1, 0 y menos 0,3, el segundo con entradas menos 4,0, 0,05 y 0,1. ¿Cuál es su producto escalar?

Multiplique las primeras entradas: 0,1 por menos 4,0 da menos 0,4.

Multiplique las segundas entradas: 0 por 0,05 da 0.

Multiplique las terceras entradas: menos 0,3 por 0,1 da menos 0,03.

Suma los tres resultados, porque el producto escalar es la suma de los productos entrada por entrada: menos 0,4 más 0 menos 0,03.

Respuesta: Menos 0,43, un solo número, ligeramente negativo, así que las dos flechas apuntan algo en contra una de otra.

Ejercicios

1. Sin calcular nada, diga si el producto escalar de dos vectores es positivo, negativo o cero cuando las dos flechas apuntan en direcciones exactamente opuestas, y por qué.

Pista: Recuerde qué hace la medida de alineación para direcciones opuestas.

Respuesta: Negativo. El producto escalar es las dos longitudes, que son positivas, por la alineación, y para direcciones opuestas la alineación está en su valor más bajo, un valor negativo.

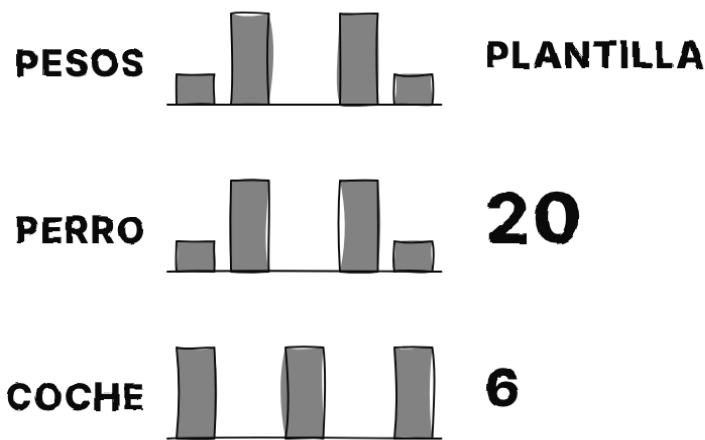
2. Explique con sus propias palabras qué permite la licencia del libro, y la única condición que lleva asociada.

Pista: Piense en compartir, cambiar, vender, y en lo que debe ocurrir con la nueva obra.

Respuesta: Cualquiera puede compartir, remezclar, adaptar y construir sobre el libro, en cualquier forma e incluso comercialmente, siempre que se dé crédito al autor. Lo que se construya sobre él debe publicarse bajo la misma licencia, por eso este curso también la lleva; las imágenes de archivo con licencia no forman parte de ese permiso y no se reutilizan.

Plantillas, lotes y máquinas hechas de números

Tome una plantilla, una cartulina con una forma recortada, y deslícela sobre una página de formas impresas. Donde la forma de la página coincide exactamente con el hueco, se ve a la perfección; donde solo coincide a medias, parte de ella queda oculta. El producto escalar se comporta igual que esa plantilla. Fije una flecha de longitud uno y pruebe contra ella todas las demás flechas de longitud uno: el producto escalar es máximo exactamente cuando la segunda flecha es igual a la primera.



Los pesos como plantilla

Este hecho tan simple sostiene buena parte del libro. En los próximos capítulos una flecha será la entrada, digamos una foto, y la otra serán los parámetros ajustables del modelo, que el libro llama pesos. El producto escalar es máximo cuando la entrada resuena con los pesos, así que los pesos actúan como una plantilla contra la que se sostiene la entrada, y el libro llama a esto comparación con plantilla. Puede imaginar una aplicación de fotos que sostiene una plantilla para perros: una foto que encaja bien con ella da un número grande, y una foto de un coche da uno pequeño.

Vienen de paso dos pequeñas observaciones. Sumar todas las entradas de un vector es lo mismo que tomar su producto escalar con un vector hecho enteramente de unos. Y la distancia entre dos vectores se puede escribir usando solo productos escalares; en particular, el producto escalar de un vector consigo mismo es su longitud al cuadrado. Los dos trucos aparecen una y otra vez, sobre el papel y en el código.

Después vienen las matrices, rejillas con cierto número de filas y columnas. El libro le pide ver una matriz como una pila de vectores, uno por fila, y llama a esa pila un lote de datos. Piense en una hoja de cálculo donde cada fila es una foto, descrita por la misma lista de números: toda la hoja es una matriz, y los modelos se escriben para trabajar con hojas enteras como esta de una vez.

1	2	0
0	1	3
2	2	1

×

1
0
2

=

1
6
4

$1 + 0 + 0 = 1$
 $0 + 0 + 6 = 6$
 $2 + 0 + 2 = 4$

TODO A LA VEZ

Una fila contra una columna

La operación clave con matrices es la multiplicación de matrices, y está construida a partir del producto escalar. Cada entrada del resultado es el producto escalar de una fila de la primera matriz con una columna de la segunda, por eso las filas de la primera deben ser exactamente tan largas como las columnas de la segunda. Cuando la segunda matriz es una sola columna, obtiene el caso especial de una matriz por un vector. Aquí está la recompensa práctica: si la primera matriz es un lote de entradas y la segunda contiene los pesos, una sola multiplicación calcula el producto escalar de cada entrada con los pesos, todo a la vez. Y un lote multiplicado por su propia versión volteada da, de una vez, el producto escalar de cada par de filas, una tabla completa de cuánto se parece cada elemento a todos los demás.

Pruébalo

Escriba una pequeña hoja de cálculo con tres filas, cada una una lista corta de tres números, y una cuarta lista de tres números en una columna aparte como sus pesos. Junto a cada fila, pida a la hoja la suma de los productos de las entradas correspondientes con los pesos. Ha calculado tres productos escalares a mano, uno por fila; una multiplicación de matrices hace exactamente esto, para miles de filas y muchas columnas de pesos en un solo paso.

Las matrices también se pueden multiplicar entrada por entrada, cuando tienen la misma forma, una regla que lleva el nombre del matemático Hadamard. Carece de las elegantes propiedades de la verdadera multiplicación de matrices, pero es extremadamente práctica: multiplicar por una rejilla de ceros y unos apaga entradas, como una máscara, y multiplicar por otros números las escala. Algunas familias de modelos recientes también se apoyan en esos productos entrada por entrada, como mostrarán capítulos posteriores.

5	2	7
1	8	3
4	6	9

 $\odot$

1	0	1
0	1	0
1	1	0

 $=$

5	0	7
0	8	0
4	6	0

MÁSCARA

Una máscara, entrada por entrada

En el mismo espíritu, cuando el libro aplica una función como la exponencial a una matriz entera, quiere decir aplicarla a cada entrada por separado. Eso no es lo mismo que la verdadera exponencial de matrices de los libros de texto, que funciona solo con matrices cuadradas y se construye a partir de multiplicaciones de matrices repetidas. La versión entrada por entrada funciona con cajas de cualquier forma, por eso bibliotecas como PyTorch guardan las operaciones genuinamente matriciales, como inversas y determinantes, en un rincón aparte de su caja de herramientas. El recuadro de código del libro muestra las dos una junto a otra, una llamada ordinaria para la exponencial entrada por entrada y una especializada para la exponencial de matrices. Por último, una reducción aplasta una caja a lo largo de una dirección, por ejemplo

sumando o promediando cada fila, y en el código se elige nombrando la dirección.



Las matrices son máquinas

Un recuadro final pregunta por qué la multiplicación de matrices se define de esta manera, fila contra columna, en lugar de entrada por entrada. La respuesta es que captura las funciones lineales, las funciones que respetan el estirar y el sumar: estire o sume las entradas primero, o las salidas después, y obtiene el mismo resultado. Toda función así se puede escribir como una matriz. Multiplicar una matriz por un vector es entonces simplemente aplicar la función, y multiplicar dos matrices es hacer una función después de la otra. Así que una matriz no es solo una tabla de números; es una pequeña máquina, y multiplicar matrices conecta máquinas entre sí.

Para recordar

Entre flechas de longitud uno, el producto escalar es máximo cuando ambas son iguales, así que los pesos de un modelo funcionan como una plantilla con la que resuena la entrada: comparación con plantilla. Una matriz se puede ver como un lote, una pila de vectores uno por fila. La multiplicación de matrices llena cada entrada con el producto escalar de una fila y una columna, así que una multiplicación calcula todos los productos escalares de un lote a la vez. La multiplicación entrada por entrada enmascara y escala, funciones como la exponencial actúan entrada por entrada, y las reducciones aplastan una caja a lo largo de una dirección. La multiplicación de matrices se define así porque una matriz es una función lineal, de modo que multiplicar por un vector la aplica y multiplicar matrices encadena funciones.

Preguntas

1. ¿Por qué el libro llama «comparación con plantilla» al producto escalar entre una entrada y los pesos?
 - a. Porque copia la entrada en los pesos
 - b. Porque es máximo cuando la entrada se alinea con los pesos, como una forma que encaja en una plantilla
 - c. Porque cuenta cuántas entradas tienen los dos vectores
 - d. Porque siempre da cero para fotos de coches

2. Un lote de entradas, una por fila, se multiplica por una matriz que contiene los pesos. ¿Qué consigue esa única multiplicación?
 - a. Promedia todas las entradas en una sola fila
 - b. Calcula el producto escalar de cada entrada con los pesos, todo a la vez
 - c. Multiplica las entradas entrada por entrada con los pesos
 - d. Eleva al cuadrado cada entrada del lote
3. Quiere apagar algunas entradas de una matriz y mantener las demás sin cambios. ¿Qué operación encaja?
 - a. La multiplicación entrada por entrada por una rejilla de ceros y unos de la misma forma
 - b. La verdadera exponencial de matrices
 - c. Multiplicar la matriz por su versión volteada
 - d. Una reducción que suma cada fila
4. ¿Por qué la multiplicación de matrices se define fila contra columna y no entrada por entrada?
 - a. Porque es más rápida de calcular en chips gráficos
 - b. Porque solo las matrices cuadradas se pueden multiplicar entrada por entrada
 - c. Porque una matriz representa una función lineal, y esta regla hace que multiplicar matrices encadene las funciones
 - d. Porque la multiplicación entrada por entrada no está permitida en PyTorch

Respuestas

1. **b)** Entre flechas de longitud uno el producto escalar es máximo cuando ambas son iguales, así que los pesos actúan como una plantilla con la que

resuena la entrada.

2. b) Cada entrada de un producto de matrices es el producto escalar de una fila con una columna, así que una multiplicación da todos los productos escalares del lote juntos.

3. a) La multiplicación de Hadamard, entrada por entrada, por ceros y unos actúa como una máscara, apagando las entradas donde la máscara tiene ceros.

4. c) Toda función lineal se puede escribir como una matriz; multiplicar por un vector la aplica y multiplicar dos matrices hace una función después de la otra.

Ejemplos resueltos

Qué formas se pueden multiplicar Un lote tiene 3 filas de 4 números cada una, y una matriz de pesos tiene 4 filas y 2 columnas. ¿Se pueden multiplicar, y qué forma tiene el resultado?

Cada entrada del producto es el producto escalar de una fila de la primera matriz con una columna de la segunda, así que deben tener la misma longitud.

Una fila del lote tiene 4 números y una columna de los pesos tiene 4 números, así que la multiplicación es posible.

Hay una entrada del resultado por cada par de una fila del lote y una columna de pesos: 3 filas emparejadas con 2 columnas.

Respuesta: Sí; el resultado tiene 3 filas y 2 columnas, una fila por entrada y una columna por conjunto de pesos.

Ejercicios

1. Sin la fórmula, explique por qué el producto escalar de un vector con un vector hecho enteramente de unos da la suma de sus entradas.

Pista: Piense en qué hace multiplicar cada entrada por uno antes de que todo se sume.

Respuesta: El producto escalar multiplica las entradas correspondientes y suma los resultados. Multiplicar cada entrada por uno la deja sin cambios, así que lo que se suma son las propias entradas, que es su suma.

2. Un lote de fotos, una por fila, se multiplica por su propia versión volteada. Describa la tabla que obtiene y qué le dice una de sus entradas.

Pista: Cada entrada empareja una fila con una fila.

Respuesta: Obtiene una tabla cuadrada con una fila y una columna por foto. Cada entrada es el producto escalar de las filas de dos fotos, una medida de cuánto se parecen esos dos elementos, así que toda la tabla compara cada elemento con todos los demás en un solo paso.

3. Explique la diferencia entre aplicar la exponencial a una matriz entrada por entrada y la verdadera exponencial de matrices, y por qué las bibliotecas las mantienen separadas.

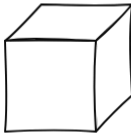
Pista: Pregúntese cuál funciona para cajas de cualquier forma.

Respuesta: Entrada por entrada, la exponencial se aplica a cada número por separado y funciona para cajas de cualquier forma. La verdadera exponencial de matrices se construye a partir de multiplicaciones de matrices repetidas y funciona solo para matrices cuadradas. Como las operaciones exclusivamente matriciales como esta, las inversas y los determinantes son especiales, bibliotecas como PyTorch las guardan en un módulo aparte.

Cuánto trabajo es una multiplicación

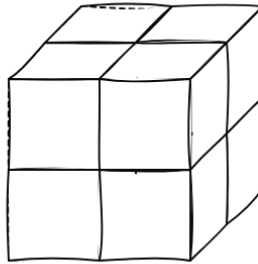
Imagine que rellena a mano una gran tabla de multiplicar, donde cada casilla exige emparejar una fila completa de una matriz con una columna completa de otra. Ahora haga ambas matrices el doble de anchas y el doble de altas, y rellene la nueva tabla. Esperaría el doble de trabajo, pero se encontrará trabajando más o menos ocho veces más tiempo. Esa sorpresa es lo que los informáticos llaman complejidad, y el libro usa la multiplicación de matrices para presentarla.

$$O(N^3)$$



N

N^3 PASOS



2N

8 x EL TRABAJO

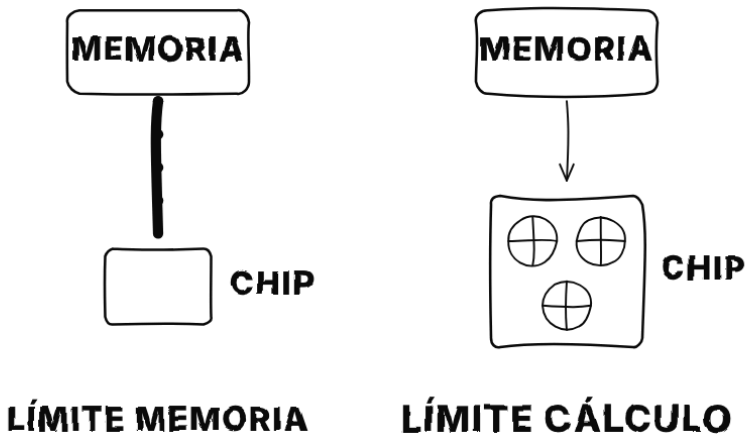
Doble tamaño, ocho veces el trabajo

Para multiplicar dos matrices directamente por la definición, calcula un producto escalar por cada casilla del resultado, y cada uno de esos productos escalares es tan largo como el lado compartido de las dos matrices. El número total de pasos se llama complejidad temporal. La memoria necesaria, si calcula las casillas una tras otra, es solo el tamaño del resultado, y eso se llama complejidad espacial.

Para hablar del coste sin atarlo a un ordenador concreto, la informática usa lo que se llama notación O grande, de la palabra alemana para orden. Describe cómo crece el coste cuando crece el problema, e ignora a propósito los factores fijos que dejan de importar cuando el problema es lo bastante grande; este tipo de razonamiento se llama análisis asintótico. En esos términos, la forma simple de multiplicar matrices crece en proporción a cada uno de los tres tamaños implicados. Para dos matrices cuadradas crece con el cubo de su tamaño, que es exactamente la razón por la que doblar el tamaño supuso ocho veces el trabajo.

El libro avisa de que esta visión elegante no cuenta toda la historia. Se conocen métodos más ingeniosos para multiplicar matrices cuadradas que crecen notablemente más despacio que el cubo, pero son difíciles de repartir entre los miles de pequeños trabajadores dentro de un chip gráfico, así que en la práctica casi nadie los usa. Lo que ocurre de verdad cuando pide a una biblioteca como PyTorch que multiplique dos matrices depende también de la máquina: la biblioteca elige una rutina de bajo nivel escrita para ese hardware, una para el procesador principal, otra para un chip gráfico, otra para chips especializados. Los compiladores pueden ir más lejos y fundir varias operaciones en una, afinada para un chip concreto. El mismo asistente de chat puede ejecutarse en un teléfono o en un centro de datos, y por debajo rutinas muy distintas hacen la multiplicación.

Conviene recordar dos límites más de la teoría. Tener muchos procesadores trabajando en paralelo solo divide el trabajo por un factor fijo, así que no cambia cómo crece el coste con el tamaño. Y la teoría ignora el tiempo que se pasa moviendo datos de un lugar de la memoria a otro, que en los programas reales es a menudo el verdadero cuello de botella. Cuando mover datos es lo que frena un programa, se dice que está limitado por memoria; cuando el límite es la propia aritmética, se dice que está limitado por cálculo. La única forma fiable de saber cuál es cuál es medir el programa en marcha con un perfilador, y el libro promete que el juego entre teoría y medición volverá una y otra vez.



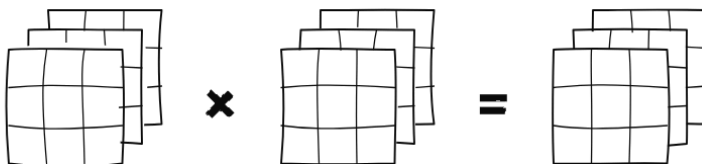
Limitado por memoria o por cálculo

Pruébalo

Piense en una cocina con un solo cocinero y la nevera en la habitación de al lado. Si cada plato necesita un corte rápido pero un paseo a la nevera por cada ingrediente, contratar más cocineros apenas ayuda: el paseo es el cuello de botella, y la cocina está limitada por memoria. Si en cambio cada plato necesita horas de cortar con todo ya en la encimera, más manos ayudan directamente: la cocina está limitada por cálculo.

Por último, el libro mira más allá de los vectores y las matrices. Muchas operaciones especiales, como los determinantes o la exponencial de matrices, solo tienen sentido para matrices, así que la mayoría de las operaciones con cajas mayores son o muchas operaciones matriciales hechas una junto a otra, o combinaciones de operaciones matriciales con reducciones. El primer tipo es la multiplicación de matrices por lotes: tome una pila de matrices y una segunda pila de la misma longitud, y multiplíquelas par a par, obteniendo una pila de resultados. Las bibliotecas tratan las primeras direcciones de una caja como direcciones de lote sin que se lo pidan, así que en PyTorch la versión por lotes se escribe exactamente igual que una multiplicación de matrices ordinaria, como muestra el recuadro de código del libro. Cuando una aplicación del teléfono procesa muchas fotos a la vez, se apoya exactamente en este hábito de manejar pilas enteras en una sola llamada.

LOTE DE 3



APLANAR, LUEGO ·

Pilas multiplicadas par a par

El segundo tipo es un producto escalar generalizado: dos cajas de la misma forma, con cualquier número de direcciones, multiplicadas entrada por entrada y después sumadas todas. No es más que aplanar cada caja en una larga fila de números y tomar un producto escalar ordinario. Con estas operaciones, dice el libro, ha cubierto la mayoría de las operaciones con tensores que hacen falta para el resto del libro.

Para recordar

Multiplicar matrices directamente cuesta un producto escalar por entrada del resultado, la complejidad temporal, y necesita memoria para el resultado, la complejidad espacial. La notación O grande describe cómo crece el coste con el tamaño, y para matrices cuadradas el método simple crece con el cubo del tamaño, así que doblar el tamaño significa unas ocho veces el trabajo. Existen métodos más rápidos sobre el papel, pero se adaptan mal a los chips paralelos, las bibliotecas eligen rutinas específicas del hardware, y muchos procesadores solo dividen el coste por un factor fijo. Mover datos es a menudo el verdadero cuello de botella, la diferencia entre programas limitados por memoria y limitados por cálculo, y solo medir dice cuál se aplica. Las cajas mayores se manejan como pilas de operaciones matriciales una junto a otra, o como productos escalares generalizados de cajas aplanadas.

Preguntas

1. Dos matrices cuadradas se hacen el doble de grandes por cada lado. ¿Cómo cambia aproximadamente el trabajo de la multiplicación simple?
 - a. Se duplica
 - b. Crece unas cuatro veces
 - c. Crece unas ocho veces
 - d. Se queda igual en un chip gráfico

2. ¿Por qué los métodos más ingeniosos para multiplicar matrices, que crecen más despacio, se usan rara vez en la práctica?
 - a. Dan respuestas erróneas para matrices grandes
 - b. Son difíciles de repartir entre los muchos pequeños trabajadores de un chip gráfico
 - c. Necesitan matrices que no sean cuadradas
 - d. Los prohíbe la notación O grande
3. Un programa apenas se acelera cuando añade más procesadores, y una medición muestra que pasa la mayor parte del tiempo trayendo datos. ¿Qué es?
 - a. Limitado por cálculo
 - b. Limitado por memoria
 - c. Asintóticamente cúbico
 - d. Libre de complejidad espacial
4. ¿Qué deja fuera a propósito la notación O grande?
 - a. Cómo crece el coste cuando crece el problema
 - b. Los factores fijos que dejan de importar cuando el problema es lo bastante grande
 - c. El tamaño de las entradas
 - d. Si una operación lleva tiempo o no

Respuestas

1. **c)** Para matrices cuadradas el método simple crece con el cubo del tamaño, así que doblar el tamaño significa unas ocho veces el trabajo.
2. **b)** Aunque sobre el papel crecen más despacio que el cubo, se paralelizan mal en hardware altamente paralelo como los chips gráficos.

3. b) Cuando lo que frena un programa es mover datos, y no la aritmética, está limitado por memoria, y solo un perfilador puede mostrarlo con fiabilidad.

4. b) La O grande describe cómo crece el coste con el tamaño e ignora los factores fijos, por eso muchos procesadores en paralelo no la cambian.

Ejemplos resueltos

La multiplicación por lotes del libro En el recuadro de código del libro, la primera pila contiene 4 matrices de 5 filas y 2 columnas, y la segunda contiene 4 matrices de 2 filas y 3 columnas. ¿Qué forma sale de la multiplicación por lotes?

La primera dirección es el lote: hay 4 pares, multiplicados par a par, así que el resultado es una pila de 4 matrices.

En cada par, una matriz de 5 por 2 se encuentra con una de 2 por 3; el lado compartido de 2 coincide, así que se pueden multiplicar.

Cada producto tiene una fila por fila de la primera matriz y una columna por columna de la segunda: 5 filas y 3 columnas.

Respuesta: Una pila de 4 matrices, cada una con 5 filas y 3 columnas.

Ejercicios

1. Explique por qué tener muchos procesadores no cambia la complejidad O grande de una operación, aunque haga el programa más rápido.

Pista: ¿Qué tipo de aceleración dan muchos procesadores, en el mejor de los casos?

Respuesta: Muchos procesadores pueden, en el mejor de los casos, dividir el trabajo por un factor fijo, el número de procesadores. La O grande ignora los factores fijos y solo describe cómo crece el coste con el

tamaño, así que el crecimiento sigue siendo el mismo aunque cada ejecución termine antes.

2. Describa con sus propias palabras el producto escalar generalizado de dos cajas con tres direcciones, y diga cómo se relaciona con el producto escalar ordinario.

Pista: Piense en desenrollar cada caja en una larga fila.

Respuesta: Dos cajas de la misma forma se multiplican entrada por entrada y todos los productos se suman en un solo número. Es exactamente el producto escalar ordinario de las dos cajas después de aplanar cada una en una larga fila de números.

Una receta en letras, y la pendiente de una curva

Imagine que da a cada dirección de una caja de números una letra, como etiquetar los cajones de un armario: una letra para qué foto, una para qué fila, una para qué color. Ahora describa una operación solo listando las letras de lo que entra y las letras de lo que debería salir. Cualquier letra que desaparezca por el camino ha sido sumada. Esa idea diminuta, tomada de una notación abreviada que Albert Einstein usaba en física, se llama einsum, y el libro la presenta como una sección opcional pero muy útil.

Tome las dos operaciones de la parte anterior. La multiplicación de matrices por lotes se convierte en una cadena corta que dice: entran dos pilas, con una letra compartida en medio, y sale una pila sin esa letra; la letra compartida es la que se suma, exactamente los productos escalares de filas y columnas. El producto escalar generalizado se convierte en una cadena aún más corta: entran dos cajas con las mismas tres letras, y no sale nada, así que todo se suma en un solo número. Hay una correspondencia directa entre estas cadenas y las definiciones escritas completas, y la mayoría de las bibliotecas ofrecen una función llamada einsum que las lee.

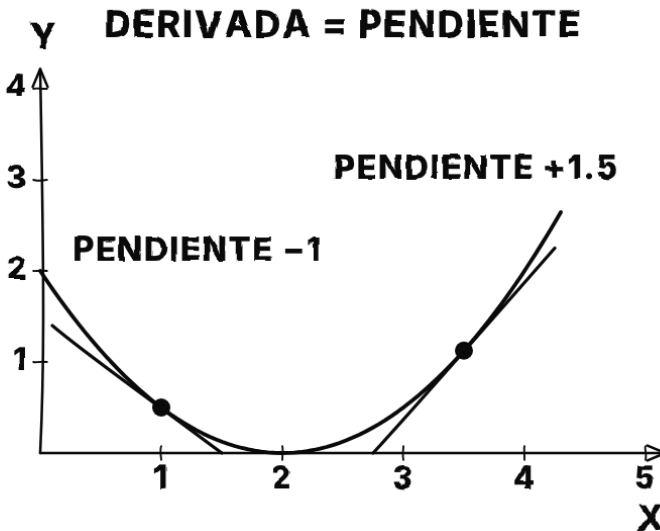
La ventaja es práctica. Ya no necesita recordar cuál de las varias funciones de multiplicación de una biblioteca tiene qué reglas de forma, porque una sola cadena las cubre todas. La misma cadena funciona sin cambios en PyTorch y en JAX, así que mover código de una biblioteca a otra se vuelve transparente. Voltar una matriz dentro de una operación es tan simple como intercambiar dos letras

en la cadena. Por estas razones einsum , y las herramientas que lo extienden como un paquete popular llamado einops , se han vuelto de uso muy extendido.

El libro pasa ahora al corazón de la palabra diferenciable: los gradientes. Son lo que permite a los modelos mejorarse a sí mismos casi automáticamente, mediante un método llamado descenso de gradiente, y el libro recuerda lo esencial, prefiriendo la utilidad al rigor matemático completo. Empieza con el caso más simple, una función que toma un número y da un número.

Pruébelo

Dibuje una curva suave que baja y después vuelve a subir, como un valle poco profundo, y elija un punto en su ladera. Coloque una regla de modo que apenas toque la curva en ese punto sin atravesarla. Ahora muévase por la curva un paso diminuto a la derecha: si la regla se inclina hacia arriba, la curva sube ahí; si se inclina hacia abajo, la curva baja. La inclinación de la regla es la derivada en ese punto.



La pendiente de una recta tangente

Esa es exactamente la imagen del libro. La derivada se define preguntando cuánto cambia la salida ante un cambio diminuto de la entrada, mientras ese cambio diminuto se encoge hacia nada. Geométricamente es la pendiente de la recta que toca la curva en un punto, y también la mejor imitación en línea recta de la función cerca de ese punto. La figura del libro muestra una curva suave en forma de valle con dos rectas tangentes así en dos puntos distintos, una inclinada hacia abajo y otra hacia arriba. Una pendiente positiva significa que, en un pequeño entorno, la función sube hacia la derecha y baja hacia la izquierda, y una pendiente negativa significa lo contrario. Es como el sensor de inclinación de su teléfono: le dice hacia dónde se inclina el suelo justo donde está parado, no lo que hay

a un kilómetro. El libro subraya que esta visión local se traslada sin cambios a las funciones de muchos números.

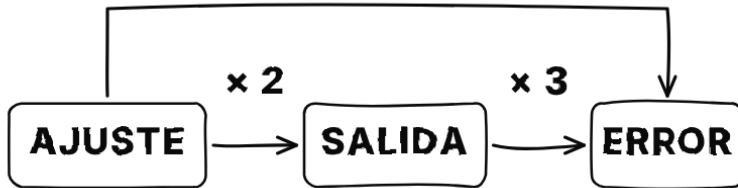
Las reglas conocidas del colegio, para potencias, logaritmos y senos, salen todas directamente de esa definición. El libro no se preocupa de si una derivada existe, lo que puede fallar incluso en una curva sin cortes, por ejemplo en la esquina afilada del valor absoluto; vuelve a esas esquinas en el capítulo seis.

Tres reglas se trasladan al caso de muchas dimensiones y se usarán constantemente. La regla de la suma dice que la pendiente de una suma es la suma de las pendientes. La regla del producto dice que cuando dos cantidades se multiplican, el cambio del producto viene de que cada una cambia mientras la otra se queda quieta, sumados. La regla de la cadena dice que cuando una función alimenta a otra, las pendientes a lo largo de la cadena se multiplican. Si un entrenamiento cambia un ajuste de un modelo, lo que cambia su respuesta, lo que cambia su error, el efecto total del ajuste sobre el error es esos efectos multiplicados a lo largo de la cadena, y esta es la regla en la que se apoyará todo el libro.

SUMA: LAS PENDIENTES SE SUMAN

PRODUCTO: CADA UNO POR TURNO

$\times 6$



$$2 \times 3 = 6$$

Las pendientes se multiplican en cadena

Para recordar

`einsum` describe casi cualquier operación con tensores como una cadena corta de letras, una por dirección, donde las letras que faltan en el resultado se suman. Una sola cadena reemplaza muchas funciones de biblioteca, funciona igual en PyTorch y JAX, y convierte el volteo en un intercambio de letras. La derivada es la pendiente de la recta que toca una curva, la mejor imitación en línea recta de la función cerca de un punto, que dice si sube o baja en las cercanías. Las reglas de la suma, del producto y de la cadena se extienden a muchas dimensiones, y la regla de la cadena, que multiplica las pendientes a lo largo de una cadena de funciones, es la que el libro usará más.

Preguntas

1. En una cadena de `einsum`, una letra aparece en las entradas pero no en la salida. ¿Qué ocurre a lo largo de esa dirección?
 - a. Se copia sin cambios en el resultado
 - b. Se suma, desaparece sumada
 - c. Se voltea, como una transposición
 - d. Provoca un error, porque toda letra debe aparecer en la salida
2. En un punto donde la pendiente de una función es negativa, ¿qué hace la función cerca de ese punto?
 - a. Baja hacia la derecha y sube hacia la izquierda

- b. Sube hacia la derecha y baja hacia la izquierda
 - c. Se mantiene plana en ambas direcciones
 - d. Debe tener una esquina afilada
3. ¿Por qué el libro encuentra einsum tan útil en la práctica?
- a. Porque es más rápido que cualquier otra multiplicación
 - b. Porque solo funciona en PyTorch
 - c. Porque una sola cadena reemplaza muchas funciones de biblioteca y funciona igual en PyTorch y JAX
 - d. Porque calcula derivadas automáticamente
4. Un ajuste de un modelo afecta a su respuesta, y la respuesta afecta a su error. ¿Cómo combina la regla de la cadena los dos efectos?
- a. Los suma
 - b. Se queda solo con el mayor
 - c. Los multiplica a lo largo de la cadena
 - d. Ignora el primero

Respuestas

- 1. b)** Por la convención de einsum, cualquier letra que desaparece en el camino de las entradas a la salida se suma.
- 2. a)** Una pendiente positiva significa subir hacia la derecha y bajar hacia la izquierda en un pequeño entorno; una pendiente negativa significa lo contrario.
- 3. c)** Una sola cadena cubre las distintas funciones de multiplicación con sus reglas de forma y se mueve sin cambios entre bibliotecas.

4. **c)** Cuando una función alimenta a otra, sus pendientes se multiplican, así que el efecto del ajuste sobre el error es el producto de los efectos a lo largo de la cadena.

Ejercicios

1. Explique con sus propias palabras por qué la derivada en un punto se llama la mejor imitación en línea recta de la función ahí, usando la regla que toca una curva.

Pista: Piense en qué hacen la regla y la curva muy cerca del punto de contacto.

Respuesta: La regla toca la curva en el punto y se inclina hacia donde se dirige la curva ahí, así que para un paso diminuto a cualquier lado la regla y la curva casi coinciden. Ninguna otra línea recta sigue la curva tan de cerca junto a ese punto, y la inclinación de esa línea es la derivada.

2. Un ajuste cambia la respuesta de un modelo tres veces más rápido de lo que se mueve el propio ajuste, y la respuesta cambia el error dos veces más rápido de lo que se mueve la respuesta. Por la regla de la cadena, ¿con qué rapidez cambia el error con el ajuste?

Pista: La regla de la cadena multiplica los efectos a lo largo de la cadena.

Respuesta: Seis veces más rápido: la regla de la cadena multiplica los dos efectos, tres por dos.

3. Describa, sin símbolos, qué dice la cadena de einsum para el producto escalar generalizado, y por qué no aparece nada después de la flecha.

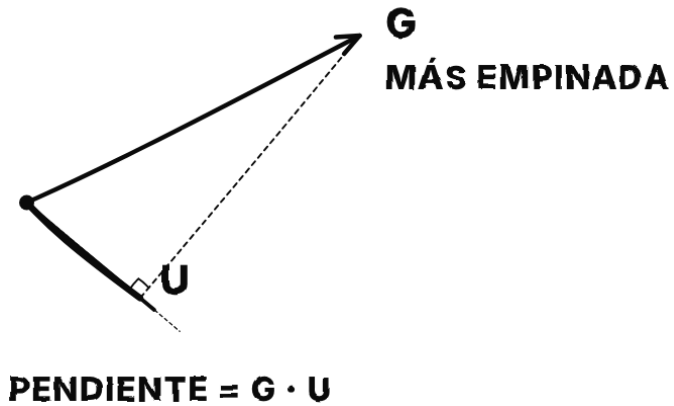
Pista: ¿Qué letras aparecen en las entradas, y cuáles en la salida?

Respuesta: Entran dos cajas con las mismas tres letras, así que se multiplican entrada por entrada. Como ninguna letra aparece en la salida, todas las direcciones se suman, y el resultado es un solo número.

Pendientes en todas las direcciones

Párese en la ladera de una colina con hierba y hágase una pregunta simple: ¿el suelo aquí sube o baja? En una colina no hay una única respuesta. Dé un paso al este y quizá suba, dé un paso al norte y quizá baje, y dé un paso al noreste y quizá se quede casi al mismo nivel. Con un solo número como entrada, una pendiente solo tenía izquierda y derecha; con muchos números como entrada hay infinitas direcciones, y el libro muestra cómo manejarlas todas.

El movimiento más simple es a lo largo de un solo eje, manteniendo fijas todas las demás coordenadas, como caminar estrictamente hacia el este. La pendiente que siente entonces se llama derivada parcial. Hágalo una vez por cada eje, este, norte y así sucesivamente, para tantas direcciones como tenga la entrada, y apile todas esas pendientes en una lista. Esa lista es el gradiente de la función, y es tan importante que el libro le da un símbolo especial propio. Puede imaginarlo como una flecha tendida sobre la ladera.



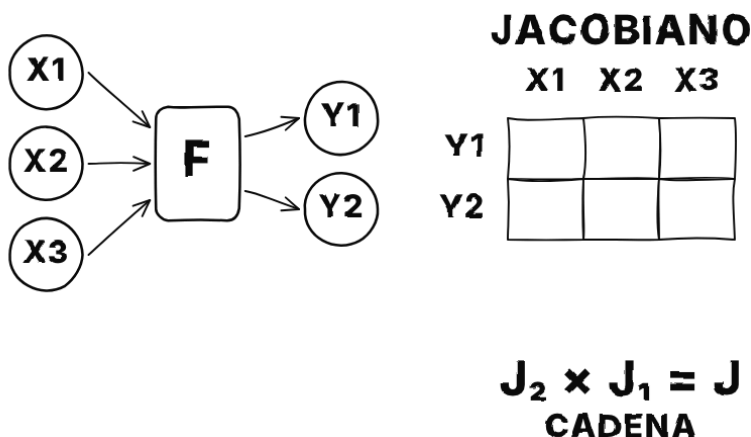
Pendiente en cualquier dirección

¿Y qué pasa al caminar en cualquier otra dirección, digamos un poco al este y mucho al norte? La pendiente a lo largo de una dirección arbitraria se llama derivada direccional. Como cualquier movimiento se puede descomponer en trozos a lo largo de cada eje, resulta ser simplemente el producto escalar del gradiente con la dirección en la que camina. Así que el gradiente solo basta: de esa única flecha puede leer la pendiente en todas las direcciones posibles. Cuando se entrena una aplicación de fotos, sus millones de ajustes son los ejes de una ladera enorme, y el gradiente dice, para todos los ajustes a la vez, hacia dónde crecen sus errores.

Pruébalo

Incline una bandeja sobre una mesa y haga rodar una canica sobre ella desde distintas direcciones de partida. Hacia donde la empuje, acaba corriendo hacia abajo por una línea concreta, la más empinada. Esa línea la fija solo la inclinación de la bandeja, y la pendiente de cualquier otra dirección es solo una parte de ella, que es exactamente lo que significa que el gradiente determina todas las pendientes direccionales.

Ahora deje que la función devuelva varios números en lugar de uno. Cada salida tiene su propio gradiente, y apilar estos gradientes, uno por fila, da una rejilla llamada jacobiano, con una fila por salida y una columna por entrada. Con una sola salida el jacobiano es simplemente el gradiente, y con una sola entrada y una sola salida es la derivada ordinaria. El libro dice que este es el caso más general que necesitará jamás.

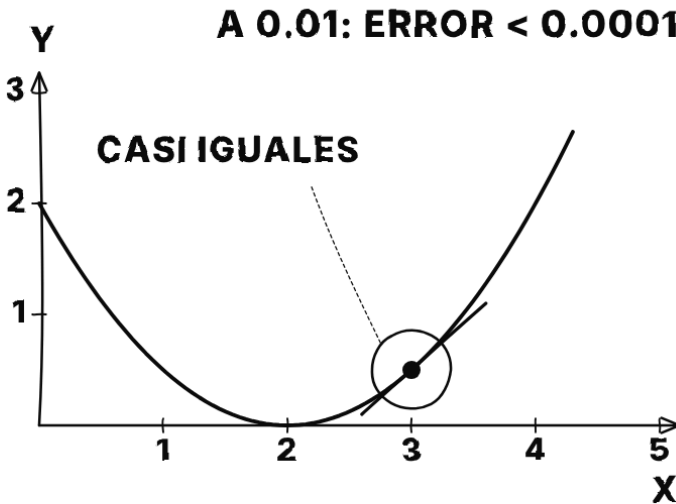


Gradientes apilados: el jacobiano

Los jacobianos conservan todas las buenas costumbres de las derivadas. Lo más importante: cuando dos funciones se encadenan, el jacobiano de la cadena es el producto de sus jacobianos, una multiplicación de matrices, que es la regla de la cadena ya mayor. Un modelo es una larga cadena de pasos, así que esta regla permite montar el efecto de cada ajuste sobre el error final multiplicando las rejillas de todos los pasos intermedios.

Los gradientes y los jacobianos también tienen un significado geométrico, como la regla tangente de antes. Cerca de un punto elegido, la función se comporta casi exactamente como una versión plana, en línea recta, de sí misma: su valor en ese punto, más el gradiente emparejado con cuánto se ha movido. Esta es la mejor

aproximación de primer orden, un resultado conocido como el teorema de Taylor. El libro lo comprueba en código sobre la misma curva en forma de valle de antes: a una centésima de paso del punto elegido, la estimación en línea recta y el valor verdadero coinciden hasta una diezmilésima.



La estimación en línea recta

Para recordar

Con muchas entradas, una pendiente depende de la dirección; moverse a lo largo de un eje da una derivada parcial, y apilarlas todas da el gradiente, una flecha. La pendiente en cualquier dirección es el producto escalar del gradiente con esa dirección, así que el gradiente dice todo sobre las pendientes locales. Con varias salidas, los gradientes se apilan en el jacobiano, una fila por salida, y las funciones encadenadas multiplican sus jacobianos. Cerca de cualquier punto una función se comporta como su aproximación en línea recta construida a partir del gradiente, que es el teorema de Taylor.

Preguntas

1. ¿Por qué conocer el gradiente basta para conocer la pendiente en todas las direcciones?
 - a. Porque una función con muchas entradas tiene una sola dirección
 - b. Porque cualquier movimiento se descompone en trozos a lo largo de los ejes, así que la pendiente es el producto escalar del gradiente con la dirección
 - c. Porque el gradiente siempre es cero
 - d. Porque la derivada direccional ignora la dirección
2. Una función toma 5 números y devuelve 3 números. ¿Qué aspecto tiene su jacobiano?

- a. Una rejilla con 3 filas y 5 columnas, una fila por salida
 - b. Una sola lista de 5 pendientes
 - c. Una rejilla con 5 filas y 3 columnas, una fila por entrada
 - d. Un solo número
3. Dos funciones se encadenan. ¿Cómo se obtiene el jacobiano de la cadena?
- a. Sumando los dos jacobianos
 - b. Multiplicando los dos jacobianos, una multiplicación de matrices
 - c. Quedándose solo con el jacobiano de la última función
 - d. Multiplicándolos entrada por entrada
4. ¿Qué dice el teorema de Taylor sobre una función cerca de un punto elegido?
- a. Que es igual a su valor en el punto en todas partes
 - b. Que se comporta casi exactamente como una versión en línea recta construida a partir de su valor y su gradiente
 - c. Que debe ser un polinomio
 - d. Que su gradiente es el mismo en todas partes

Respuestas

- 1. b)** La derivada direccional es igual al producto escalar del gradiente con la dirección del movimiento, así que el gradiente solo da todas las pendientes direccionales.
- 2. a)** Cada salida tiene su propio gradiente, uno por fila, así que el jacobiano tiene una fila por salida y una columna por entrada.

3. b) La regla de la cadena crece hasta convertirse en una multiplicación de matrices de los jacobianos de las funciones encadenadas.

4. b) Cerca del punto la función se imita mejor con su valor más el gradiente emparejado con el desplazamiento: la aproximación de primer orden.

Ejemplos resueltos

La comprobación en línea recta del libro El libro toma la curva en forma de valle $f(x) = x^2 - 1.5x$, cuya pendiente es $2x - 1.5$, construye su aproximación en línea recta en $x = 0.5$ y la compara con la curva verdadera en $x = 0.51$. ¿Qué valores obtiene?

Valor en el punto elegido: 0.5 al cuadrado es 0.25 , menos 1.5 por 0.5 , que es 0.75 , da menos 0.5 .

Pendiente en el punto elegido: 2 por 0.5 menos 1.5 da menos 0.5 .

Estimación en línea recta una centésima a la derecha: menos 0.5 más la pendiente por 0.01 , es decir menos 0.5 menos 0.005 , que es menos 0.505 .

Valor verdadero: 0.51 al cuadrado es 0.2601 , menos 1.5 por 0.51 , que es 0.765 , da menos 0.5049 .

Respuesta: La estimación es menos 0.5050 y el valor verdadero menos 0.5049 : difieren solo en una diezmilésima.

Ejercicios

1. Explique con sus propias palabras la diferencia entre una derivada parcial y una derivada direccional, usando la ladera.

Pista: Una trata de caminar a lo largo de un solo eje; la otra, de cualquier dirección elegida.

Respuesta: Una derivada parcial es la pendiente que siente cuando camina estrictamente a lo largo de un eje, digamos hacia el este, manteniendo fijas todas las demás coordenadas. Una derivada direccional

es la pendiente a lo largo de cualquier dirección que elija, por ejemplo un poco al este y mucho al norte; combina las derivadas parciales según cuánto se mueve a lo largo de cada eje.

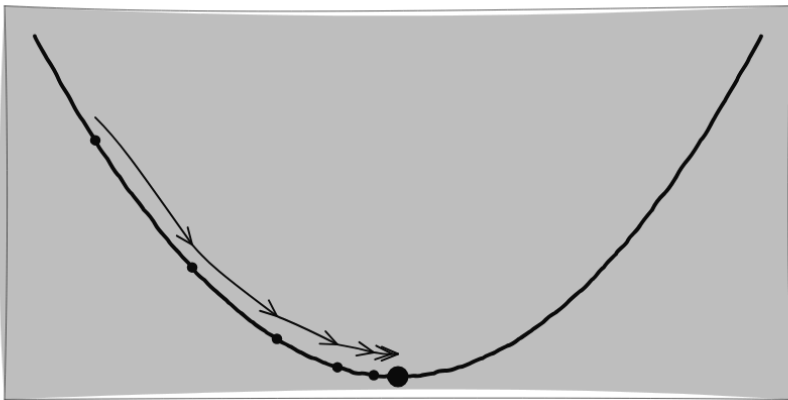
2. ¿Por qué importa tanto la regla de la cadena para jacobianos en los modelos, según esta parte?

Pista: Piense en un modelo como una larga cadena de pasos.

Respuesta: Un modelo es una larga cadena de pasos, y el efecto de cada ajuste sobre el error final pasa por todos los pasos intermedios. Como el jacobiano de una cadena es el producto de los jacobianos de sus pasos, ese efecto se puede montar multiplicando las rejillas de todos los pasos.

Bajar la montaña en la niebla

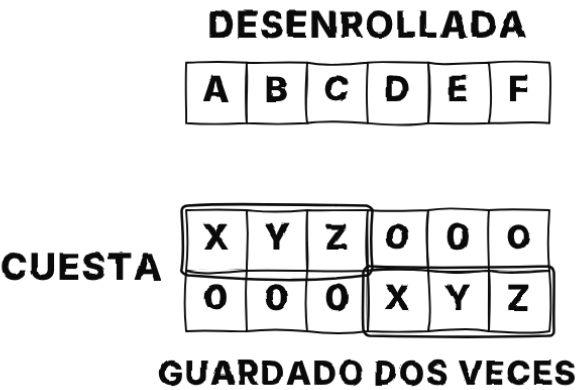
Imagine que está en una montaña con niebla espesa y quiere llegar al fondo del valle. No puede ver el valle, solo el suelo justo bajo sus pies. Así que tantea hacia dónde cae el suelo con más pendiente, da un pequeño paso en esa dirección y vuelve a tantear. Paso a paso, sin ver nunca el paisaje completo, se va abriendo camino hacia abajo. Eso es el descenso de gradiente, el método en el centro de esta parte, pero antes el libro añade una nota cuidadosa sobre los tamaños de los jacobianos.



MÍNIMO

Tanteando el camino hacia abajo

Tome el paso de modelo más simple: una rejilla de pesos multiplicada por un vector de entrada. Visto como función de la entrada, su jacobiano es simplemente la propia rejilla de pesos, lo cual tranquiliza: la pendiente de una máquina lineal es la máquina. Visto como función de los pesos, la entrada de la función es ella misma una rejilla, y la pendiente se convierte en una caja con tres direcciones, una por cada salida y dos para la posición de cada peso. El truco del libro es imaginar la rejilla de pesos desenrollada en una larga fila de números; entonces la pendiente vuelve a ser una rejilla ordinaria. Por eso el libro puede escribir todo con vectores sin perder generalidad.

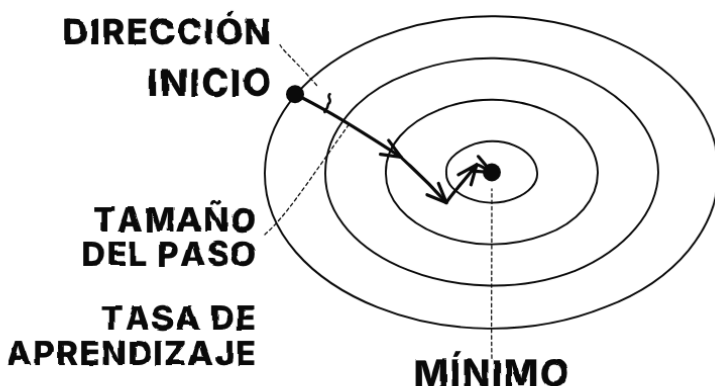


Desenrollar la rejilla de pesos

Un pequeño recuadro lo resuelve entrada por entrada. Cada salida es una suma de pesos por entradas, así que la pendiente de una salida

con respecto a uno de sus pesos es simplemente la entrada correspondiente. Escribir toda la caja de pendientes en memoria significaría, por tanto, guardar los mismos números de la entrada una y otra vez. El libro promete que en el capítulo seis este desperdicio desaparece, porque en la práctica nunca necesitamos la caja en sí, solo lo que hace cuando se aplica a otra caja. También avisa de que las rejillas pulcras escritas sobre el papel pueden esconder tamaños enormes dentro, sobre todo el tamaño del lote, así que el coste real de memoria puede ser mucho mayor de lo que sugiere la notación. Un lote de entrenamiento de miles de fotos hace enormes esas cajas escondidas, por eso el software de entrenamiento real nunca las construye.

Ahora el plato principal: encontrar el punto más bajo de una función, su mínimo. El libro supone que la función da un solo número y que la entrada puede vagar por cualquier parte. En el resto del libro la entrada serán los parámetros del modelo y la función medirá lo mal que lo hace el modelo con los datos, el planteamiento del aprendizaje supervisado del capítulo siguiente. Minimizar no es menos general que maximizar: ponga la curva boca abajo y sus picos más altos se convierten en sus valles más profundos.

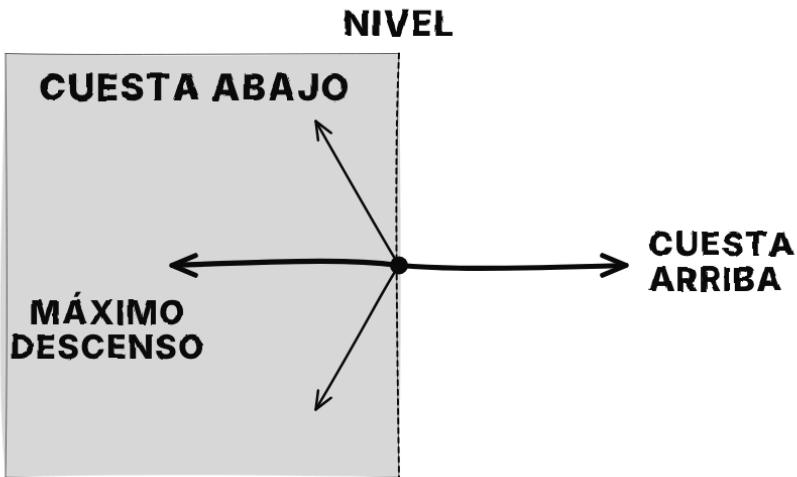


Paso a paso hasta el fondo

Solo en casos raros el punto más bajo se puede escribir directamente con una fórmula; el libro mostrará un caso así más adelante, para ajustar líneas rectas. Normalmente debemos buscar paso a paso. Empezamos desde una suposición al azar y damos un paso repetidamente, que tiene dos ingredientes: su longitud, llamada tamaño del paso, y hacia dónde apunta, su dirección. En aprendizaje automático el tamaño del paso se llama tasa de aprendizaje, por razones que el capítulo siguiente aclara. Una dirección que puede bajar la función, con un paso lo bastante pequeño, se llama dirección de descenso, y si cada paso usa una y los tamaños de paso se eligen con cuidado, la búsqueda se asienta en un mínimo.

Pruébalo

Ponga una bola sobre una tabla inclinada y dele pequeños empujones en distintas direcciones. Empújela recta hacia arriba por la pendiente y sube; empújela de lado a lo largo de la pendiente y se queda a la misma altura; empújela en cualquier dirección que apunte aunque sea un poco hacia abajo y desciende. Empújela recta hacia abajo por la pendiente y desciende más rápido que de ninguna otra forma.



Qué direcciones van hacia abajo

Eso es exactamente lo que nos dice el gradiente, a través de la pendiente direccional de la parte anterior. La pendiente en una

dirección es el producto escalar del gradiente con esa dirección, lo que significa la longitud del gradiente por lo bien alineada que está la dirección con él. La longitud del gradiente no depende de la dirección que elijamos, y podemos tomar toda dirección con longitud uno, así que solo decide la alineación. Cualquier dirección que se aparte del gradiente más de un ángulo recto va hacia abajo, y la que apunta exactamente en sentido opuesto al gradiente baja con la mayor pendiente de todas: la dirección de máximo descenso.

Junte las dos ideas y tiene un algoritmo para minimizar cualquier función diferenciable, el descenso de gradiente de máxima pendiente. Desde la suposición actual, dé un pequeño paso en contra del gradiente, después calcule el gradiente de nuevo en el nuevo lugar, y repita. El libro no se ocupa de elegir el tamaño del paso; simplemente supone que los pasos son lo bastante pequeños para que cada uno baje la función. Entrenar el modelo detrás de un asistente de chat es, en el fondo, este mismo paseo repetido una cantidad inmensa de veces sobre un paisaje con miles de millones de direcciones. El descenso de gradiente es solo tan rápido como nuestra forma de calcular gradientes, y un método general para eso es el tema del capítulo seis.

Para recordar

Visto desde la entrada, el jacobiano de una rejilla de pesos por un vector es la propia rejilla de pesos; visto desde los pesos, es una caja mayor que se puede aplanar en una rejilla ordinaria. Esas cajas repiten muchos valores y esconden tamaños de lote, así que los métodos prácticos solo las aplican, nunca las guardan. Para minimizar una función normalmente buscamos paso a paso, y cada paso tiene una longitud, la tasa de aprendizaje, y una dirección. Cualquier dirección que se aparte del gradiente más de un ángulo recto va hacia abajo, y la exactamente opuesta al gradiente baja más rápido. El descenso de gradiente repite pequeños pasos en contra del gradiente, y su velocidad depende de lo rápido que se puedan calcular los gradientes.

Preguntas

1. Entre todas las direcciones de longitud uno, ¿cuál hace que una función caiga más rápido?
 - a. La dirección del gradiente
 - b. Cualquier dirección en ángulo recto con el gradiente
 - c. La dirección exactamente opuesta al gradiente
 - d. Todas las direcciones caen igual de rápido
2. ¿Por qué dice el libro que podemos estudiar la minimización sin perder nada sobre la maximización?
 - a. Porque las funciones nunca tienen puntos máximos

- b. Porque poner una función boca abajo convierte sus picos en valles
 - c. Porque el descenso de gradiente solo puede encontrar puntos máximos
 - d. Porque maximizar no necesita gradiente
3. ¿Por qué los métodos prácticos evitan guardar la caja completa de pendientes de una rejilla de pesos por una entrada?
- a. Porque repetiría los mismos valores de la entrada muchas veces y escondería tamaños de lote grandes
 - b. Porque siempre es cero
 - c. Porque no se puede calcular en absoluto
 - d. Porque tiene menos entradas que la entrada
4. ¿Qué decide lo rápido que puede funcionar el descenso de gradiente en la práctica, según el libro?
- a. El color de los datos
 - b. Lo rápido que se pueden calcular los gradientes
 - c. Si la función está escrita con letras
 - d. Cuántas veces se pone la curva boca abajo

Respuestas

- 1. c)** Solo la alineación con el gradiente decide la pendiente, y apuntar exactamente en su contra da la pendiente más negativa: la dirección de máximo descenso.
- 2. b)** Minimizar una función es lo mismo que maximizar su versión boca abajo, así que un solo método cubre ambas cosas.
- 3. a)** Cada pendiente con respecto a un peso es simplemente la entrada correspondiente, así que la caja está llena de repeticiones, y los tamaños de

lote pueden hacerla enorme; solo se necesita su efecto sobre otra caja.

4. b) El descenso de gradiente es solo tan eficiente como el procedimiento para calcular gradientes, el tema del capítulo seis.

Ejercicios

1. Un caminante en una pendiente prueba una dirección que forma con el gradiente un ángulo menor que un recto. ¿El caminante va cuesta abajo o cuesta arriba, y por qué?

Pista: Recuerde qué significa la alineación con el gradiente para la pendiente.

Respuesta: Cuesta arriba. Una dirección a menos de un ángulo recto del gradiente está alineada positivamente con él, así que la pendiente en esa dirección es positiva; solo las direcciones a más de un ángulo recto van cuesta abajo.

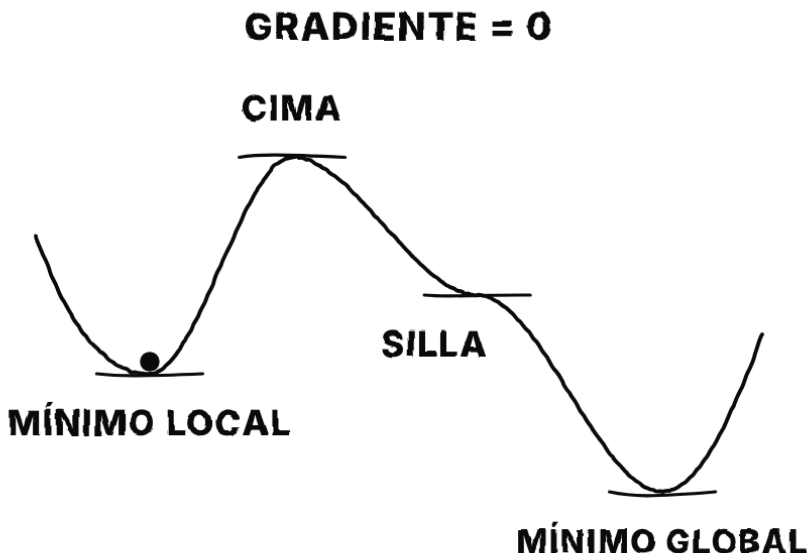
2. Describa el descenso de gradiente con sus propias palabras, nombrando los dos ingredientes de cada paso y lo que ocurre entre pasos.

Pista: Piense en el caminante en la niebla.

Respuesta: Empiece desde una suposición. Cada paso tiene una longitud, el tamaño del paso o tasa de aprendizaje, y una dirección, que para el máximo descenso es exactamente opuesta al gradiente. Tras el paso, el gradiente se calcula de nuevo en el nuevo lugar y el proceso se repite, con pasos lo bastante pequeños para que cada uno baje la función.

Dónde puede terminar el paseo

Vierta un poco de agua sobre una hoja arrugada de papel de aluminio y observe dónde se asienta. No corre toda hacia el único punto más bajo de la hoja; se junta en la pequeña hondonada que resulta estar más cerca. El descenso de gradiente se comporta igual, y el libro usa esto para hacer una pregunta precisa: cuando el paseo cuesta abajo se detiene, ¿dónde exactamente se ha detenido? El autor añade que si simplemente confía en el descenso de gradiente, puede saltar adelante; esta parte es para quienes quieren saber qué garantiza.



Lugares planos en una curva arrugada

Un mínimo local es un punto más bajo que todo lo que hay en un pequeño entorno a su alrededor, el fondo de una hondonada del papel de aluminio. En un punto así la recta tangente es plana, y a su alrededor el suelo sube. La planitud la captura una idea más amplia, el punto estacionario: cualquier punto donde el gradiente es cero, donde no hay pendiente alguna.

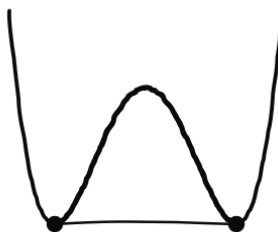
No todo punto plano es una hondonada. La cima de una colina también es plana, y también lo es un punto de silla, un lugar donde el suelo se curva hacia arriba en una dirección y hacia abajo en otra, como el centro de la silla de un caballo o un paso de montaña. La figura del libro muestra una silla así, e invita a imaginar ahí la recta tangente plana. Sin más conocimiento de la función, lo único que se puede demostrar es que el descenso de gradiente termina en algún punto estacionario, y cuál depende de dónde empezó.

Algunos paisajes son más amables. Piense en un cuenco simple, como un valle tallado en una sola curva suave: no tiene sillars y solo un fondo, y ese fondo es el punto más bajo de todo el paisaje, un mínimo global. Desde cualquier punto de partida, toda pendiente en un cuenco lleva a ese fondo, así que el descenso de gradiente no puede fallarlo. El libro generaliza el cuenco con la idea de una función convexa.

CONVEXA



NO CONVEXA



La prueba del hilo

Pruébalo

Dibuje cualquier curva en forma de valle, elija dos puntos en ella y tienda un hilo recto entre ellos. Si la curva entre los dos puntos se queda siempre por debajo del hilo, elija los dos puntos que elija, la curva es convexa. Ahora dibuje una curva con dos hondonadas y un bulto entre ellas y pruebe de nuevo: con los dos puntos a cada lado del bulto, la curva sube por encima del hilo, y no es convexa.

La convexidad le dice lo fácil que es la búsqueda. En un paisaje no convexo, el descenso de gradiente termina en algún punto

estacionario, y no se puede decir nada más sin mirar cómo cambian las propias pendientes. En un paisaje convexo, alcanza un mínimo global desde cualquier punto de partida. Y si la curva se queda estrictamente por debajo de todo hilo, el mínimo global es además único. Para los problemas no convexos la mala noticia es grave: para estar seguro del mínimo global tendría que reiniciar la búsqueda desde cada punto de partida posible, lo que hace la tarea irremediabilmente difícil en general.

Esto tiene un profundo peso histórico. Como mostrará el capítulo cinco, cualquier modelo que merezca la pena usar es no convexo, así que su paisaje de entrenamiento puede estar lleno de hondonadas y sillars. Otros métodos de aprendizaje, como las máquinas de vectores de soporte, resultaban atractivos precisamente porque mantienen una búsqueda convexa y segura mientras manejan aun así patrones que no son rectos. Sin embargo, los modelos diferenciables complejos funcionan notablemente bien en la práctica: iniciado desde un punto sensato, el entrenamiento tiende a terminar en lugares que rinden bien con datos reales. El asistente de chat que usted usa fue entrenado en un paisaje arrugado así, y funciona aunque nadie pueda prometer que su entrenamiento encontró el punto más bajo de todos.

Para recordar

Un mínimo local es más bajo que sus alrededores cercanos, y es un punto estacionario, un lugar con gradiente cero, como las cimas y los puntos de silla. En general, al descenso de gradiente solo se le garantiza detenerse en algún punto estacionario, según dónde empiece. Una función convexa se queda por debajo de la recta entre dos cualesquiera de sus puntos, como un cuenco, y ahí el descenso de gradiente alcanza un mínimo global desde cualquier parte, único si la convexidad es estricta. Los modelos útiles son no convexos, a diferencia de las máquinas de vectores de soporte, pero en la práctica su entrenamiento aterriza en puntos que funcionan bien.

Preguntas

1. El descenso de gradiente se detiene donde el gradiente es cero. ¿Por qué eso no garantiza que haya encontrado un mínimo?
 - a. Porque el gradiente nunca es exactamente cero
 - b. Porque un gradiente cero también ocurre en las cimas y en los puntos de silla
 - c. Porque los mínimos siempre tienen un gradiente grande
 - d. Porque el descenso de gradiente solo funciona en cuencos
2. En una función convexa, ¿qué se puede decir del descenso de gradiente?
 - a. Alcanza un mínimo global desde cualquier punto de partida

- b. Siempre se queda atascado en un punto de silla
 - c. Debe reiniciarse desde todos los puntos posibles
 - d. Solo funciona si se empieza en el mínimo
3. ¿Por qué resultaban atractivas las máquinas de vectores de soporte frente a las redes neuronales?
- a. No necesitan datos
 - b. Mantienen una búsqueda convexa y segura mientras manejan aun así patrones que no son rectos
 - c. Siempre encuentran puntos de silla
 - d. Se entrenan con pasos más grandes

Respuestas

1. **b)** Cualquier punto con gradiente cero es un punto estacionario, y los puntos estacionarios incluyen máximos y puntos de silla además de mínimos.
2. **a)** Como un cuenco, una función convexa lleva toda pendiente a su punto más bajo, así que el descenso de gradiente alcanza un mínimo global desde cualquier parte.
3. **b)** Los modelos neuronales útiles son no convexos, mientras que las máquinas de vectores de soporte manejan patrones no lineales manteniendo convexa la optimización.

Ejercicios

1. Explique con sus propias palabras la diferencia entre un mínimo local y un mínimo global, usando el papel de aluminio arrugado.

Pista: Uno trata de un pequeño entorno; el otro, de toda la hoja.

Respuesta: Un mínimo local es el fondo de una hondonada: más bajo que todo lo que está cerca, pero no necesariamente el punto más bajo de la hoja. Un mínimo global es el punto más bajo de toda la hoja. El agua vertida sobre el aluminio suele asentarse en la hondonada más cercana, un mínimo local, no necesariamente el global.

2. ¿Por qué es tan difícil en general encontrar el mínimo global de una función no convexa con descenso de gradiente?

Pista: Piense en cómo el punto de partida decide dónde termina el paseo.

Respuesta: En un paisaje no convexo, dónde termina el paseo depende de dónde empieza, y puede detenerse en cualquier hondonada o lugar plano. Para estar seguro del punto más bajo de todos tendría que empezar desde cada punto posible, lo que en general es irremediablemente caro.

3. A pesar de ser no convexas, ¿por qué las redes neuronales siguen funcionando bien, según el libro?

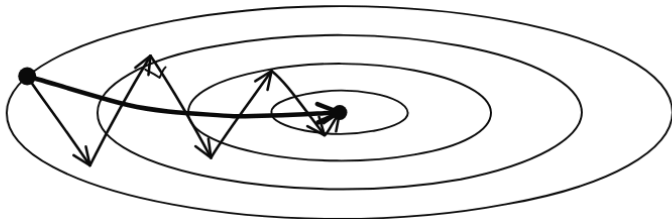
Pista: Piense en lo que ocurre cuando el entrenamiento empieza desde un punto sensato.

Respuesta: En la práctica, cuando el entrenamiento empieza desde una inicialización razonable, el descenso de gradiente tiende a terminar en puntos que rinden bien con datos reales, aunque nadie pueda garantizar que sean el mínimo global.

Una bola con impulso

Haga rodar una canica por una sábana arrugada. Una canica que obedeciera solo a la pendiente bajo ella en cada instante se sacudiría de un lado a otro en cada arruga. Una canica real no lo hace: conserva parte de su velocidad y su dirección de un momento antes, y rueda con suavidad sobre los pequeños bultos. Esa memoria del movimiento pasado, el momento o impulso, es el primer truco que el libro muestra para acelerar el descenso de gradiente.

SIN MEMORIA



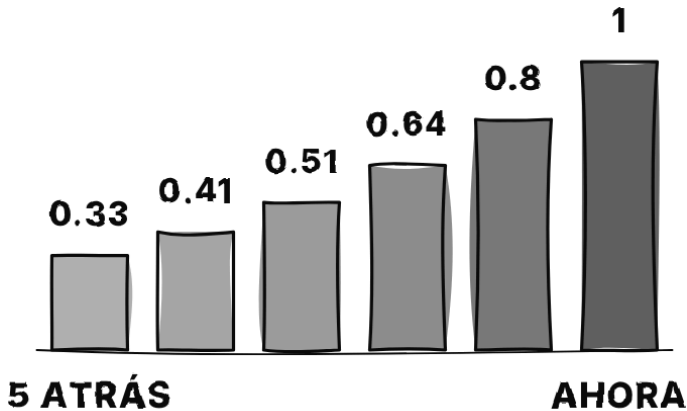
MOMENTO

Zigzag frente a momento

El opuesto del gradiente es el camino más empinado hacia abajo, pero solo en un entorno diminuto del punto actual. El libro avisa de que, como mostrará el capítulo cinco, estas direcciones pueden ser muy ruidosas, sobre todo en modelos grandes. Se han inventado muchos métodos para elegir mejores direcciones y llegar al fondo más deprisa. Al libro le interesan sobre todo los que no necesitan ni pendientes de pendientes, que son caras, ni muchas evaluaciones extra de la función.

Imagine el descenso de gradiente simple como una bola que rueda cuesta abajo sin memoria: su camino puede ser errático, porque cada nuevo gradiente puede apuntar a un lugar bastante distinto. El libro incluso señala que, con un tamaño de paso ideal sobre una función en forma de cuenco, dos gradientes consecutivos forman un ángulo recto entre sí, así que el camino zigzaguea. El momento suaviza el zigzag conservando una parte del paso anterior y sumándola al nuevo. Un solo ajuste decide cuánto del pasado se conserva. Desenrollar la regla muestra que los pasos más antiguos siguen contando, pero cada uno un poco menos que el siguiente, desvaneciéndose cuanto más atrás queda.

GUARDAR 0.8 DEL PASADO



Los pasos antiguos cuentan menos

La figura del libro compara los dos sobre una curva ondulada, el descenso de gradiente simple y la versión con momento, ambos iniciados desde el mismo punto. Se sabe que el momento acelera el entrenamiento al suavizar el camino. Otro truco habitual da a cada parámetro su propio tamaño de paso, adaptado a lo grandes que han sido sus gradientes. Un método muy usado llamado Adam combina varias de estas ideas.

Pruébelo

Llene un cuenco de agua y remuévala con golpecitos diminutos que cambian de dirección sin parar: el agua apenas se mueve. Ahora remueva con firmeza en una sola dirección durante unos segundos: el agua empieza a girar y sigue girando incluso después de que usted pare. Los empujones pequeños que están de acuerdo entre sí se suman, y los empujones que se cancelan continuamente no: eso es lo que el momento hace con los pasos del descenso de gradiente.

La gran virtud práctica de Adam es que funciona bien sin mucho trasteo con sus ajustes. Los ajustes que elige el usuario, en lugar de aprenderse por descenso de gradiente, se llaman hiperparámetros, y para Adam los valores por defecto de la biblioteca suelen ser un buen comienzo. Muchos modelos que encuentra cada día, de las aplicaciones de fotos a los asistentes de chat, se han entrenado con Adam o con un pariente cercano. Construir un nuevo optimizador que pueda reemplazar a Adam, o a su variante AdamW, como opción por defecto sigue siendo un problema de investigación abierto. El precio de estas aceleraciones es la memoria: el momento, por ejemplo, debe recordar el paso anterior, lo que duplica la memoria que necesita el propio optimizador, aunque calcular el gradiente suele necesitar bastante más memoria todavía.

El capítulo termina con una sección llamada «De la teoría a la práctica». En lugar de ejercicios clásicos, el libro propone un camino autoguiado por dos bibliotecas de programación, JAX y PyTorch, con soluciones en la página web del libro, y avisa de que los enlaces web pueden haberse movido cuando usted los lea. El bloque de partida es

NumPy, la biblioteca estándar para trabajar con cajas de números: crearlas, cortar rebanadas de ellas y transformarlas. El autor recomienda familiarizarse con ella, sobre todo con el corte de rebanadas, y señala una colección pública de cien ejercicios suaves para ponerse a prueba.

Para recordar

La dirección más empinada vale solo muy localmente y puede ser ruidosa, así que los métodos más rápidos eligen mejores direcciones sin derivadas extra caras. El momento conserva una parte que se desvanece de los pasos pasados, suavizando el camino en zigzag del descenso de gradiente simple y acelerando el entrenamiento. Adam combina el momento con un tamaño de paso para cada parámetro, y funciona bien con sus hiperparámetros por defecto, así que sigue siendo la opción estándar. Estos trucos cuestan memoria, y el camino práctico empieza por aprender bien NumPy.

Preguntas

1. ¿Qué añade el momento al descenso de gradiente simple?
 - a. Una parte del paso anterior, sumada al nuevo, de modo que el camino se suaviza
 - b. Una segunda derivada calculada en cada paso
 - c. Un salto aleatorio a un nuevo punto de partida
 - d. Un tamaño de paso mayor para cada parámetro

2. Con momento, ¿cuánto sigue contando un paso dado hace mucho?
 - a. Exactamente tanto como el paso más nuevo
 - b. Nada en absoluto después de un paso
 - c. Un poco menos con cada paso más atrás, desvaneciéndose
 - d. Más que el paso más nuevo
3. ¿Por qué es Adam tan popular como optimizador por defecto?
 - a. No necesita gradientes en absoluto
 - b. Funciona bien sin mucho trasteo, con los valores por defecto de la biblioteca como buen comienzo
 - c. No usa memoria más allá de los parámetros
 - d. Siempre encuentra el mínimo global
4. ¿Cuál es el coste principal de usar momento?
 - a. Hace el camino más errático
 - b. Debe recordar el paso anterior, duplicando la memoria del optimizador
 - c. Exige que la función sea convexa
 - d. Deja de funcionar en modelos grandes

Respuestas

1. **a)** El momento conserva una parte del paso anterior y la suma al nuevo, suavizando el zigzag del descenso de gradiente simple.
2. **c)** Desenrollar la regla muestra que los pasos más antiguos siguen contando, cada uno un poco menos que el siguiente, desvaneciéndose cuanto más atrás queda.

3. b) Adam es relativamente robusto a la elección de sus hiperparámetros, y los valores por defecto de la mayoría de las bibliotecas son un buen punto de partida.

4. b) El momento guarda el paso anterior, lo que duplica la memoria del optimizador, aunque calcular el gradiente suele necesitar mucha más memoria.

Ejercicios

1. Usando el cuenco de agua removida, explique por qué el momento ayuda cuando los gradientes sucesivos cambian de dirección sin parar.

Pista: Piense en qué pasa con los empujones que están de acuerdo y con los que se cancelan.

Respuesta: Los empujones en direcciones cambiantes se cancelan en gran parte, así que con momento las partes en zigzag de los pasos se encogen, mientras que las partes que coinciden de un paso a otro se suman. El camino se vuelve más suave y avanza con más firmeza cuesta abajo.

2. ¿Qué es un hiperparámetro, y en qué se diferencia de los parámetros que aprende el descenso de gradiente? Ponga como ejemplo el ajuste del momento.

Pista: ¿Quién lo elige: el usuario o el entrenamiento?

Respuesta: Un hiperparámetro es un ajuste que el usuario elige antes del entrenamiento, mientras que los parámetros los ajusta el descenso de gradiente. La parte de los pasos pasados que conserva el momento es un hiperparámetro: lo elige el usuario, y no se aprende.

Del código de práctica a los datos reales

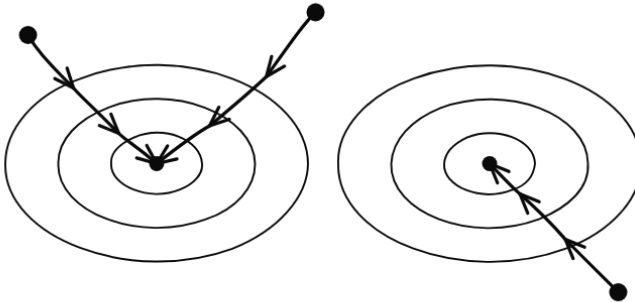
Imagine que intenta escribir a mano una lista de reglas que distinga un coche de una bicicleta en cualquier foto: el color, el contorno, el número de ruedas, las sombras, el ángulo. Nunca terminaría. Sin embargo, reunir diez mil fotos etiquetadas como coche o bicicleta es fácil. Ese contraste, reglas imposibles pero ejemplos fáciles, es el punto de partida del capítulo tres, pero antes el libro termina sus consejos prácticos.

NumPy, con toda su influencia, tiene dos debilidades para nuestros fines: por sí solo aprovecha poco los chips gráficos, y no puede calcular gradientes automáticamente. JAX copia la forma de trabajar de NumPy y añade soporte para ese hardware, gradientes automáticos y transformaciones extra, como una que convierte una función escrita para una sola entrada en una que maneja un lote entero. PyTorch también tiene un núcleo parecido a NumPy, con algunos cambios de nombres y de comportamiento, más herramientas de alto nivel para construir modelos. El autor sugiere hojear las páginas básicas sobre arrays de ambas bibliotecas para ver cuánto comparten con NumPy, e ignorar por ahora las partes de alto nivel para construir modelos.

Después viene un ejercicio práctico, para hacer tres veces, una en cada biblioteca, que debería llevar solo minutos una vez que conoce la sintaxis. Usa una función de dos números con varias colinas y valles, y el autor añade una nota franca: esa única función la generó ChatGPT, y nada más en el libro fue escrito por un modelo de lenguaje, una aclaración que el autor siente que se está volviendo importante. Primero escribe la función de modo que acepte un lote entero de

puntos a la vez, después escribe su gradiente a mano, ya que los gradientes automáticos llegan solo más adelante. A continuación ejecuta el descenso de gradiente desde varios puntos de partida y dibuja los caminos, y por último añade momento y observa cómo la longitud del gradiente se encoge hacia cero mientras la búsqueda alcanza un lugar plano.

EL INICIO DECIDE EL VALLE



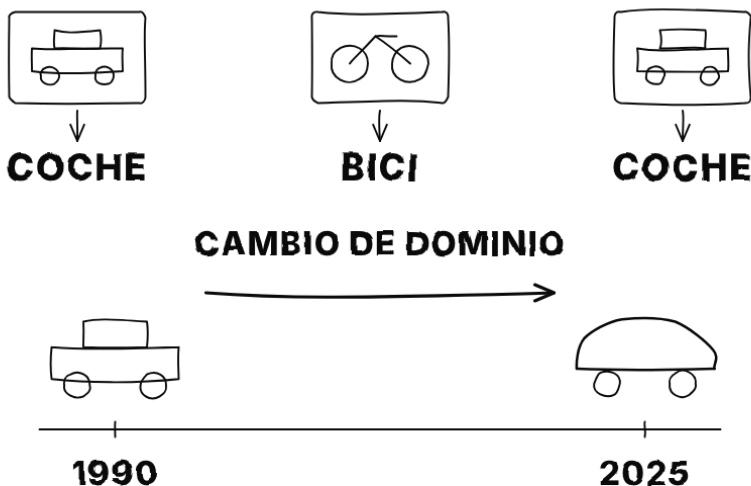
Distintos inicios, distintos valles

Pruébalo

Si tiene un ordenador a mano, pruebe primero el espíritu del ejercicio sobre papel. Dibuje un paisaje de unas pocas colinas y valles como curvas de nivel, como un mapa de senderismo, y marque tres puntos de partida. Desde cada uno, dibuje flechas cortas que apunten siempre recto cuesta abajo a través de las curvas de nivel: verá que distintos puntos de partida terminan en distintos valles, igual que mostrarán las gráficas del ejercicio.

El capítulo tres hace ahora preciso el aprendizaje supervisado: conjuntos de datos, pérdidas, la idea de minimizar el error medio sobre los datos, las suposiciones detrás de todo esto y una versión construida sobre la probabilidad. El autor llama a este capítulo corto la columna vertebral del resto del libro. Empieza exactamente donde nos dejó el problema del coche y la bicicleta: tareas que no se pueden programar a mano pero para las que es fácil reunir ejemplos, desde reconocer el habla hasta guiar un robot. Un conjunto de datos es entonces simplemente una colección de pares, cada uno una entrada junto con la salida que queremos para ella.

CONJUNTO DE DATOS: ENTRADA → RESPUESTA



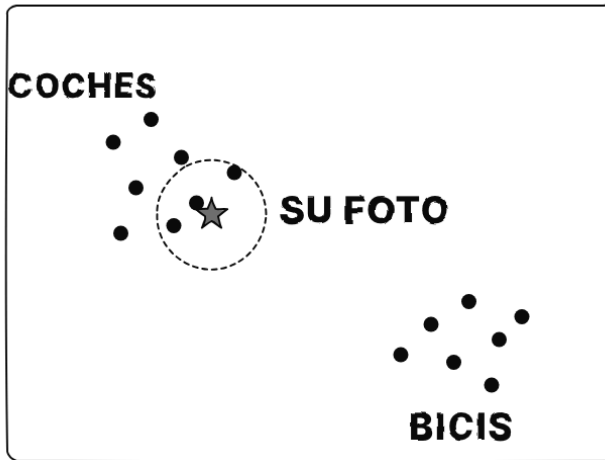
Pares de un mundo que cambia

Una suposición sobre cómo se reunieron los pares importa enormemente, y el libro la explica con coches. Los ejemplos deben extraerse de la misma manera de una fuente estable, que no cambia. Si reúne fotos de coches durante unos meses, eso se cumple; si las reúne durante varias décadas, el diseño de los coches cambia, y un modelo entrenado con ellas fallará con los modelos nuevos o funcionará mal. También deben reunirse sin sesgo, para que la colección represente con justicia todo lo que el modelo encontrará. Fotos tomadas solo cerca del concesionario de una marca estarían llenas de esa marca y escasas de todas las demás. Y la respuesta depende del contexto: fotos de coches de Italia pueden servir para un modelo usado en Roma o Milán, pero no para uno usado en Tokio.

El libro insiste en que esto no se comprueba una vez y se olvida; hay que vigilarlo durante toda la vida de un modelo. Si los coches de la carretera cambian poco a poco y nadie lo nota, el rendimiento del modelo se degrada en silencio, un problema llamado cambio de dominio. Un sistema de recomendación, como el que elige lo que sus aplicaciones le sugieren a continuación, incluso cambia el propio comportamiento del que aprende, porque la gente reacciona a sus sugerencias, creando bucles de retroalimentación que exigen una comprobación constante. Para los enormes modelos de lenguaje de hoy, entrenados con datos tan vastos y variados, se vuelve difícil incluso decir qué tareas cuentan como conocidas y cuáles como nuevas.

La figura del capítulo añade una imagen que vale la pena conservar. Un modelo entrenado transforma su entrada paso a paso, y las versiones internas de los datos que produce tienen a menudo una geometría con sentido: imágenes de significado parecido caen en puntos cercanos de un espacio interno. Convertir datos en bruto, como fotos, en puntos donde la distancia significa similitud se llama hacer un embedding de los datos. Es la razón por la que su aplicación de fotos puede encontrar imágenes parecidas a una que elija: las fotos parecidas están juntas en su espacio interno.

EMBEDDING



Lo parecido queda cerca

Para recordar

JAX y PyTorch conservan la forma de trabajar de NumPy pero añaden soporte para chips gráficos y gradientes automáticos, y el ejercicio del libro implementa el descenso de gradiente sobre una función con colinas en las tres bibliotecas. El aprendizaje supervisado es para tareas que no podemos programar a mano pero sí mostrar con ejemplos, y un conjunto de datos es una colección de pares entrada-salida. Los ejemplos deben venir de una fuente estable y reunirse sin sesgo, lo que depende del contexto y debe comprobarse durante toda la vida de un modelo para detectar el cambio de dominio y los bucles de retroalimentación. Los modelos entrenados a menudo hacen un embedding de los datos de modo que los elementos parecidos quedan juntos en un espacio interno.

Preguntas

1. Un modelo de reconocimiento de coches se entrena con fotos reunidas a lo largo de los últimos cuarenta años. ¿Qué problema predice el libro?
 - a. El modelo será demasiado pequeño para guardar tantas fotos
 - b. El diseño de los coches cambió con el tiempo, así que la fuente no era estable y el modelo puede fallar con los modelos nuevos
 - c. Las fotos serán demasiado parecidas entre sí
 - d. El modelo solo reconocerá bicicletas

2. ¿Por qué las fotos tomadas solo cerca del concesionario de una marca formarían un mal conjunto de datos?
 - a. Porque se reunirían con sesgo y no representarían todos los coches que el modelo encontrará
 - b. Porque las fotos de concesionarios siempre salen borrosas
 - c. Porque las fotos vendrían de un periodo demasiado largo
 - d. Porque un conjunto de datos debe contener también bicicletas
3. ¿Qué significa que un modelo entrenado hace un embedding de sus datos?
 - a. Esconde los datos para que nadie pueda leerlos
 - b. Convierte los datos en bruto en puntos donde la cercanía significa similitud
 - c. Guarda cada foto dos veces
 - d. Quita todos los números de los datos
4. ¿Por qué hay que comprobar las suposiciones sobre un conjunto de datos durante toda la vida de un modelo, y no solo una vez?
 - a. Porque las bibliotecas cambian de nombre cada año
 - b. Porque el mundo que encuentra el modelo puede cambiar poco a poco, y algunos sistemas incluso cambian el comportamiento del que aprenden
 - c. Porque los conjuntos de datos pierden ejemplos con el tiempo
 - d. Porque los gradientes crecen después del entrenamiento

Respuestas

1. **b)** Los ejemplos deben venir de una fuente estable, que no cambia; a lo largo de décadas el diseño de los coches cambia, así que el modelo puede fallar con los modelos nuevos o rendir mal.
2. **a)** Los ejemplos deben reunirse sin sesgo; cerca de un concesionario, esa marca estaría sobrerrepresentada y faltarían las demás marcas y modelos.
3. **b)** Dentro de un modelo entrenado, los elementos de significado parecido caen en puntos cercanos, lo que se llama hacer un embedding de los datos.
4. **b)** Los cambios lentos causan cambio de dominio, y sistemas como los recomendadores crean bucles de retroalimentación al cambiar cómo se comporta la gente.

Ejercicios

1. El libro le pide implementar el descenso de gradiente sobre una función con colinas de dos números en NumPy, JAX y PyTorch. Enumere los cuatro pasos del ejercicio y diga qué espera ver en el último.

Pista: Piense en la función apta para lotes, el gradiente escrito a mano, los caminos y el momento.

Respuesta: Primero, escribir la función de modo que maneje un lote entero de puntos a la vez; segundo, escribir su gradiente a mano; tercero, ejecutar el descenso de gradiente desde varios puntos de partida y dibujar los caminos; cuarto, añadir momento y graficar la longitud del gradiente. En el último paso la longitud del gradiente debería encogerse hacia cero mientras la búsqueda se acerca a un lugar plano.

2. Un conjunto de datos de coches reunido en Italia se usa para un modelo desplegado en Milán y otro desplegado en Tokio. Explique por qué las suposiciones pueden cumplirse para uno y no para el otro.

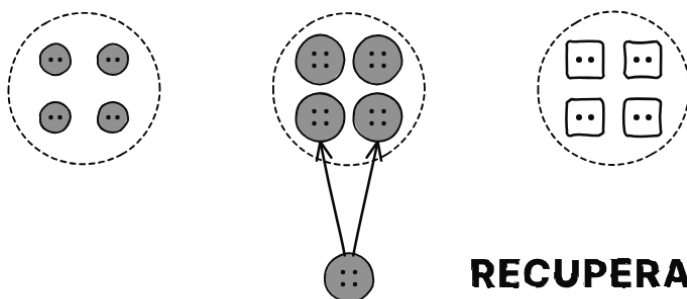
Pista: Pregúntese si los coches que el modelo encuentra se parecen a los coches del conjunto de datos.

Respuesta: Los coches de las carreteras de Milán se parecen a los fotografiados en otras partes de Italia, así que el conjunto de datos representa lo que ese modelo encontrará. En Tokio la mezcla de marcas y modelos es distinta, así que el conjunto de datos ya no representa los coches que ve el modelo y su rendimiento puede resentirse.

Aprender sin etiquetas, y reutilizar lo aprendido

Vuelque una caja de botones mezclados sobre una mesa y ordénelos en montones, sin que nadie le diga cuáles deben ser los montones. Probablemente los agrupará por color, por tamaño o por número de agujeros, y los montones tendrán sentido aunque nadie le haya dado ni una sola etiqueta. El aprendizaje supervisado siempre tenía una respuesta para cada ejemplo; esta parte mira qué pasa cuando no hay respuestas, y cómo un modelo entrenado una vez puede reutilizarse muchas veces.

AGRUPAMIENTO

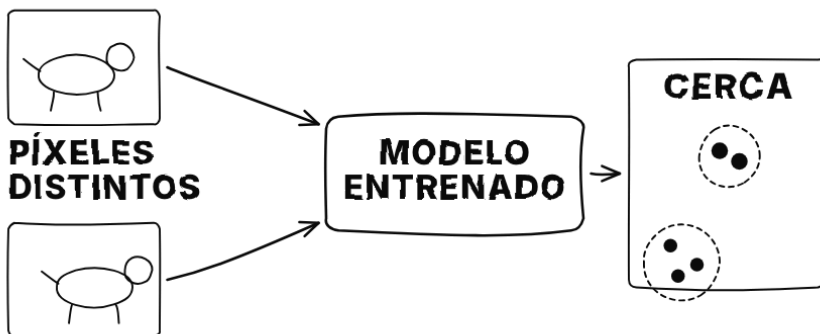


RECUPERACIÓN

Montones sin etiquetas

Cuando un conjunto de datos tiene entradas pero no respuestas objetivo, el libro habla de aprendizaje no supervisado. Una tarea clásica es el agrupamiento, o clustering: agrupar los datos de modo que los elementos dentro de un montón sean parecidos y los de montones distintos no lo sean, como los botones. Otra es la recuperación: buscar en una colección enorme los pocos elementos más parecidos a lo que alguien pidió. Pedir a su aplicación de fotos que encuentre más imágenes como una que seleccionó es exactamente una recuperación así.

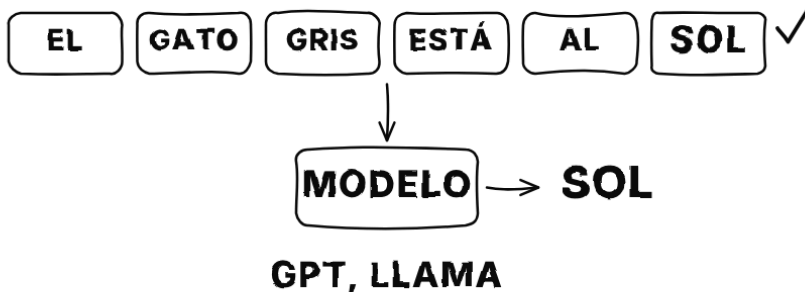
Con imágenes, sin embargo, la similitud es sorprendentemente difícil. Comparar fotos píxel a píxel falla, porque un desplazamiento diminuto de la cámara cambia millones de píxeles mientras la imagen significa lo mismo. La salida del libro es un modelo preentrenado, uno ya entrenado para una tarea bastante general, como nombrar lo que hay en una foto. Sus estados internos se pueden leer como puntos en un espacio con muchísimas direcciones, y a menudo las cosas parecidas caen ahí juntas, como vimos con los embeddings. Así que puede tomar esos puntos internos y ejecutar sobre ellos un método de agrupamiento ordinario, y los montones siguen ahora el significado en lugar de los píxeles.



Los píxeles discrepan, el significado coincide

¿Y si no hay ningún modelo preentrenado disponible? Entonces una variante popular llamada aprendizaje autosupervisado inventa una tarea supervisada a partir de datos sin etiquetar. Con una gran colección de texto, por ejemplo, siempre puede entrenar un programa para adivinar cómo continúa un trozo corto de texto, porque la continuación verdadera está ahí mismo, en el texto. Descubrir que las redes entrenadas así también aprenden embeddings excelentes del texto tuvo un impacto profundo en el campo. El libro añade una advertencia: el texto a escala de la web está lleno de sesgos y contenido grosero, los modelos absorben esos sesgos, y esta es una de las críticas principales a los modelos fundacionales cerrados.

LA RESPUESTA ESTABA EN EL TEXTO



Una tarea escondida en el texto

Los grandes modelos de lenguaje de hoy, como GPT o Llama, son la forma moderna de esta idea: su entrenamiento siempre empieza adivinando el siguiente trozo de texto, y están entre los modelos fundacionales vistos antes. La figura del libro muestra tres formas de usar un modelo así. El uso sin ejemplos, zero-shot, simplemente le hace al modelo una pregunta nueva directamente. El prompt con pocos ejemplos, few-shot, pone primero en el prompt un puñado de ejemplos resueltos de la nueva tarea, y ambos funcionan solo si el modelo generaliza bien. El ajuste fino va más lejos y ajusta los parámetros del modelo por descenso de gradiente sobre un pequeño conjunto de datos para la nueva tarea, muy parecido a entrenar desde cero, solo que más corto. Cuando pega dos respuestas de muestra en

un asistente de chat y pide una tercera en el mismo estilo, está haciendo un prompt con pocos ejemplos.

El ajuste fino es fácil hoy gracias a las grandes colecciones abiertas de modelos compartidos. Puede ajustar todos los parámetros, o solo una pequeña parte, o unos pocos añadidos, una familia de técnicas llamada ajuste fino eficiente en parámetros, que el autor deja para el siguiente volumen. Cuando un modelo aprende una tarea solo a partir de ejemplos en su prompt, sin ningún ajuste fino, el libro lo llama aprendizaje en contexto. Existen muchas más variantes: conjuntos de datos etiquetados solo en parte, que el capítulo doce toca; conjuntos de datos relacionados o los mismos datos cambiando con el tiempo; y problemas como adaptarse a nuevos dominios, aprender a aprender, aprender de forma continua, aprender distancias, e incluso hacer que un modelo olvide. El libro solo insinúa estos temas y deja varios para su segundo volumen.

Para recordar

Sin respuestas objetivo, el aprendizaje no supervisado agrupa los datos en clústeres o recupera los elementos más parecidos. Para imágenes, las distancias entre píxeles fallan, pero las representaciones internas de un modelo preentrenado colocan juntas las cosas parecidas, así que el agrupamiento puede funcionar sobre ellas. El aprendizaje autosupervisado inventa su propia tarea, como adivinar cómo continúa un texto, y así es como los modelos de lenguaje como GPT y Llama empiezan su entrenamiento, sesgos incluidos. Un modelo fundacional puede usarse sin ejemplos, con unos pocos ejemplos en el prompt, o ajustarse por descenso de gradiente, por completo o con métodos eficientes en parámetros.

Preguntas

1. ¿Por qué comparar dos fotos píxel a píxel es una mala forma de juzgar si se parecen?
 - a. Porque las fotos tienen muy pocos píxeles
 - b. Porque un desplazamiento diminuto de la cámara cambia millones de píxeles mientras el significado sigue igual
 - c. Porque los píxeles no se pueden guardar como números
 - d. Porque las fotos parecidas siempre tienen píxeles idénticos
2. ¿Cómo puede el aprendizaje autosupervisado entrenar con un montón de texto sin ninguna etiqueta?

- a. Pidiendo a la gente que etiquete cada frase
 - b. Inventando una tarea cuya respuesta ya está en el texto, como adivinar cómo continúa un trozo de texto
 - c. Agrupando el texto en montones al azar
 - d. Ajustando un modelo que ya existe
3. Da a un asistente de chat dos ejemplos de una tarea en su prompt y después pide un tercero, sin cambiar el modelo. ¿Cómo se llama esto?
- a. Ajuste fino
 - b. Agrupamiento
 - c. Prompt con pocos ejemplos
 - d. Ajuste fino eficiente en parámetros
4. ¿Qué distingue el ajuste fino del uso sin ejemplos y con pocos ejemplos?
- a. El ajuste fino cambia los parámetros del modelo por descenso de gradiente sobre un pequeño conjunto de datos
 - b. El ajuste fino no necesita datos en absoluto
 - c. El ajuste fino solo funciona con datos no supervisados
 - d. El ajuste fino es lo mismo que hacer la pregunta directamente

Respuestas

- 1. b)** Pequeñas perturbaciones cambian cantidades enormes de píxeles, así que las distancias entre píxeles dicen poco sobre el significado; las representaciones internas de un modelo preentrenado funcionan mejor.
- 2. b)** La continuación verdadera está ahí mismo, en el texto, así que predecirla da una tarea supervisada gratis.

- 3. c)** Poner un puñado de ejemplos en el prompt es un prompt con pocos ejemplos; no se cambia ningún parámetro, lo que el libro llama aprendizaje en contexto.
- 4. a)** El uso sin ejemplos y con pocos ejemplos deja el modelo tal como está, mientras que el ajuste fino ajusta sus parámetros con ejemplos de la nueva tarea.

Ejercicios

- 1.** Tiene miles de fotos de vacaciones sin etiquetas, y quiere agruparlas según lo que muestran. Describa cómo las ideas del libro le permiten hacerlo.

Pista: Los píxeles no bastan; piense en dónde vive la similitud con sentido.

Respuesta: Pase cada foto por un modelo preentrenado, por ejemplo uno entrenado para nombrar objetos, y tome su representación interna, un punto en un espacio de muchas dimensiones donde las cosas parecidas caen juntas. Después ejecute un método de agrupamiento ordinario sobre esos puntos, para que los grupos sigan el significado en lugar de los píxeles.

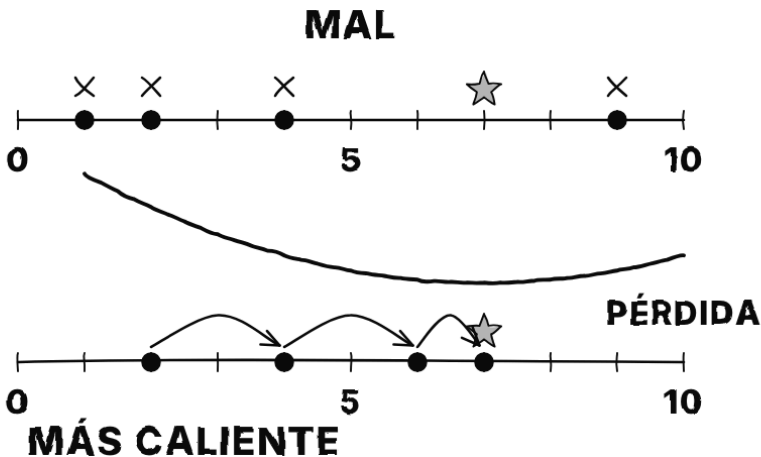
- 2.** Explique con sus propias palabras la advertencia del libro sobre entrenar con texto a escala de la web.

Pista: Piense en lo que contiene el texto y en lo que el modelo aprende de él.

Respuesta: El texto de la web contiene sesgos y contenido grosero. Un modelo entrenado para imitar ese texto absorbe esos sesgos, lo que es una de las críticas principales a los modelos fundacionales cerrados cuyos datos de entrenamiento no se pueden inspeccionar.

Puntuar una suposición

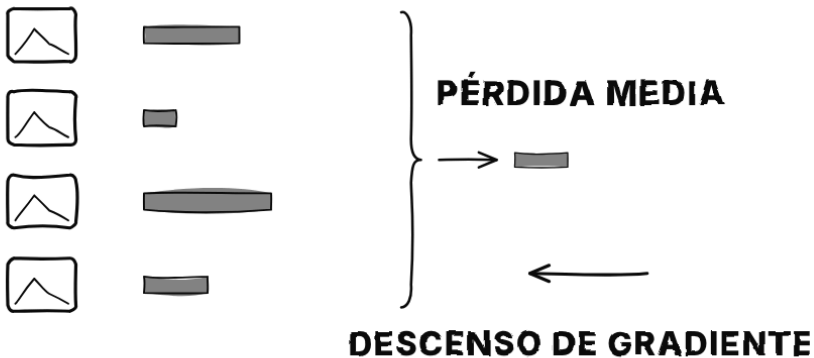
Juegue a las adivinanzas con un amigo: él piensa un número, usted adivina, y él solo responde «bien» o «mal». Intento tras intento oye «mal», y no tiene ni idea de si se está acercando. Ahora cambie la regla: él le dice «más caliente» o «más frío». De repente cada intento le enseña hacia dónde moverse, y da con el número rápidamente. Esa diferencia es toda la historia de esta parte: un modelo solo puede aprender si sus errores se puntúan de una forma que dice hacia dónde moverse.



¿Mal, o más caliente?

La puntuación se llama función de pérdida. Compara la predicción del modelo con la respuesta objetivo y da un solo número, más bajo cuanto mejor, de modo que una pérdida menor siempre significa una predicción mejor. Y debe ser diferenciable, para que podamos saber cómo cambia cuando se mueve un poco la predicción. El libro subraya que una función de pérdida empaqueta nuestra comprensión de la tarea y nuestras preferencias entre las respuestas posibles en una sola escala con la que un optimizador puede trabajar.

EJEMPLOS PÉRDIDA



Pérdida media, empujada hacia abajo

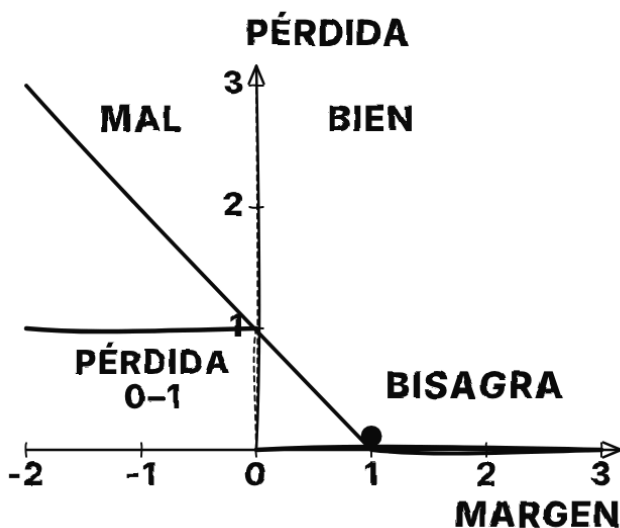
El entrenamiento se convierte entonces en una tarea matemática clara: encontrar el modelo cuya pérdida, promediada sobre todos los ejemplos del conjunto de datos, sea lo más pequeña posible. Por razones históricas esto se llama minimización del riesgo empírico,

donde riesgo es simplemente otra palabra para pérdida, y empírico significa medido sobre los datos que de verdad tenemos. En la práctica no buscamos entre todas las funciones imaginables; el modelo tiene una forma fija con parámetros ajustables, y el descenso de gradiente afina esos parámetros para bajar la pérdida media. Cuando se entrena una aplicación de fotos, esto es literalmente lo que ocurre: sus parámetros se mueven una y otra vez para bajar su puntuación media de error sobre millones de fotos etiquetadas.

Pruébalo

Esconda una moneda bajo uno de una fila de vasos y pida a alguien que la encuentre, respondiendo primero solo «sí» o «no» a cada intento, y después diciendo a cuántos vasos de distancia está. Con sí o no, debe comprobar vaso tras vaso al azar. Con «a qué distancia», puede ir directo hacia ella. Lo primero es una pérdida que nunca dice hacia dónde ir; lo segundo es una pérdida con pendiente.

El libro muestra por qué la diferenciabilidad no es un tecnicismo. Tome una tarea con dos respuestas posibles, digamos sí o no, y un modelo que responde con el signo de un número que calcula. La puntuación más natural es simplemente bien o mal: cero para una respuesta correcta, uno para una incorrecta. Coincide con nuestro sentido de lo correcto, pero es inútil para entrenar: casi en todas partes su pendiente es cero, así que el descenso de gradiente no recibe ninguna pista y se queda atascado donde empezó.



Un escalón no tiene pendiente

Una cantidad menos intuitiva pero más suave es el margen, que compara el número del modelo con el objetivo: positivo cuando el signo coincide con la respuesta correcta, negativo cuando no, y cada vez más positivo cuanto más seguro y acertado está el modelo. Como el margen cambia con suavidad, una pérdida construida sobre él da pendientes útiles. Una pérdida así, la pérdida de bisagra, usada para entrenar máquinas de vectores de soporte, es cero cuando el margen es cómodamente grande y crece de forma constante cuando el margen se encoge o se vuelve erróneo. El ejemplo muestra una tensión duradera: una pérdida debe capturar lo que entendemos por buen rendimiento y, al mismo tiempo, ser cómoda para la optimización numérica.

Un pequeño recuadro remonta estas ideas al informático ruso Vladimir Vapnik y al campo de la teoría del aprendizaje estadístico. Ese campo estudia lo bien que la pérdida media sobre un conjunto de datos limitado sustituye a la media sobre todos los datos posibles, para familias de modelos bien comportadas. Las redes neuronales modernas lo han desconcertado: siguen generalizando bien mucho después de que la teoría dijera que deberían empezar a memorizar, y esto ha abierto nuevas líneas de investigación.

Para recordar

Una función de pérdida puntúa una predicción frente a su objetivo con un solo número, más bajo cuanto mejor, y debe ser diferenciable para decir hacia dónde ajustar. El entrenamiento es la minimización del riesgo empírico: afinar los parámetros del modelo por descenso de gradiente para hacer la pérdida media sobre el conjunto de datos lo más pequeña posible. Una simple puntuación de bien o mal es inútil para entrenar, porque su pendiente es casi siempre cero. Cantidades suaves como el margen, y pérdidas construidas sobre ellas como la pérdida de bisagra, dan pendientes que seguir, lo que muestra la tensión entre lo que entendemos por bueno y lo que se puede optimizar.

Preguntas

1. ¿Por qué una simple puntuación de bien o mal es inútil para entrenar con descenso de gradiente?

- a. Porque da números demasiado grandes
 - b. Porque su pendiente es cero casi en todas partes, así que el descenso de gradiente no recibe ninguna pista y se queda atascado
 - c. Porque no se puede promediar sobre un conjunto de datos
 - d. Porque premia las respuestas incorrectas
2. ¿Qué pide la minimización del riesgo empírico?
- a. El modelo con la menor pérdida media sobre los ejemplos del conjunto de datos
 - b. El modelo con más parámetros
 - c. El modelo que nunca comete un solo error
 - d. El modelo con el mayor margen en un solo ejemplo
3. Un modelo responde sí o no con el signo de un número que calcula. ¿Cuándo es negativo su margen?
- a. Cuando el modelo acierta y está muy seguro
 - b. Cuando el signo del número del modelo no coincide con la respuesta correcta
 - c. Cuando el número del modelo es exactamente cero
 - d. Nunca; los márgenes siempre son positivos
4. ¿Qué enigma sobre las redes neuronales modernas menciona el libro en relación con la teoría del aprendizaje estadístico?
- a. No se pueden entrenar con descenso de gradiente
 - b. Siguen generalizando bien mucho después de que la teoría esperara que empezaran a memorizar
 - c. Solo funcionan con la puntuación de bien o mal
 - d. Nunca usan funciones de pérdida

Respuestas

1. **b)** La puntuación de bien o mal solo cambia cuando la respuesta se da la vuelta, así que casi en todas partes no tiene pendiente que seguir.
2. **a)** El entrenamiento se convierte en encontrar los parámetros que hacen la pérdida, promediada sobre el conjunto de datos, lo más pequeña posible.
3. **b)** El margen es positivo cuando el signo coincide con el objetivo y negativo cuando no, y cambia con suavidad.
4. **b)** Las redes muestran una fuerte generalización mucho más allá del punto donde se esperaba el sobreajuste, abriendo nuevas direcciones de investigación.

Ejercicios

1. Usando el juego de la moneda bajo los vasos, explique por qué una función de pérdida debe ser diferenciable para que el descenso de gradiente funcione.

Pista: Compare las respuestas de sí o no con las respuestas de a qué distancia.

Respuesta: Con solo sí o no, un intento fallido no dice nada sobre hacia dónde moverse, así que quien busca debe probar vasos al azar. Con a qué distancia, cada intento dice qué dirección reduce la distancia. Una pérdida diferenciable es como la respuesta de a qué distancia: su pendiente dice al descenso de gradiente hacia dónde ajustar los parámetros.

2. Describa con palabras el comportamiento de la pérdida de bisagra, y explique por qué es más útil que la puntuación de bien o mal.

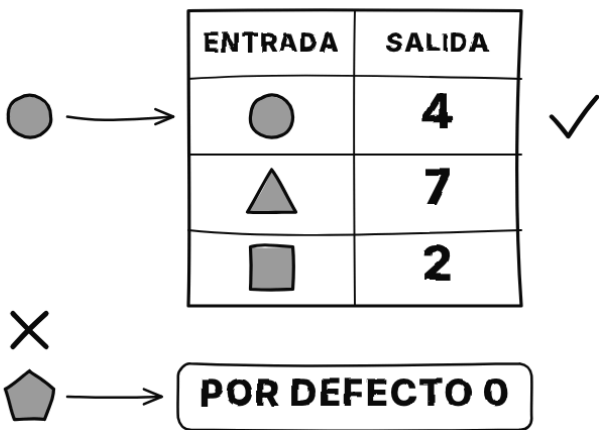
Pista: ¿Qué pasa cuando el margen es cómodamente grande, y cuando se encoge o se vuelve erróneo?

Respuesta: La pérdida de bisagra es cero cuando el margen es cómodamente grande, es decir, cuando el modelo acierta con claridad, y crece de forma constante cuando el margen se encoge o se vuelve negativo. Como cambia con suavidad con el margen, tiene pendientes que dicen al descenso de gradiente cómo mejorar, a diferencia de la puntuación de bien o mal.

Memorizar no es aprender

Imagine a un estudiante que consigue el examen del año pasado, memoriza todas las respuestas y saca la nota máxima en un ensayo con ese mismo examen. Después llega el examen real con preguntas nuevas, y el estudiante no tiene nada que decir. La nota del ensayo fue perfecta y no significaba nada. El libro usa exactamente esta trampa para mostrar por qué una pérdida media baja sobre los datos de entrenamiento no es la meta real.

ENTRENAMIENTO: SIN ERRORES

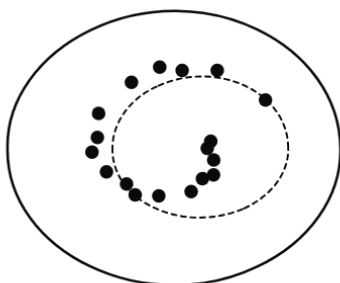


Una tabla que solo recuerda

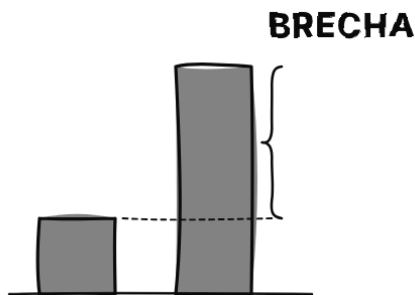
De hecho, la pérdida media sobre los datos de entrenamiento siempre es trivialmente fácil de llevar a su mínimo. Construya una tabla de consulta: si la entrada es uno de los ejemplos de entrenamiento, devuelva su respuesta guardada; si no, devuelva algún valor fijo por defecto, como cero. Esta tabla nunca se equivoca con los datos de entrenamiento, así que alcanza la pérdida media más baja posible, y sin embargo es completamente inútil en la práctica. Esa es la diferencia entre memorizar y aprender.

Lo que de verdad queremos es una pérdida baja en entradas futuras que nadie ha visto todavía; los ejemplos de entrenamiento son solo un sustituto de ellas. El libro lo llama el riesgo esperado: la pérdida media sobre todos los pares entrada-salida posibles que el modelo pueda encontrar jamás, ponderados por lo probable que es cada uno, por ejemplo sobre todos los correos electrónicos posibles. Un modelo con un riesgo esperado bajo tendría garantizado funcionar bien, pero no se puede calcular, porque nadie puede listar y etiquetar todas las entradas posibles. La pérdida media sobre el conjunto de datos, el riesgo empírico de antes, es una estimación de él, construida a partir de las muestras que resulta que tenemos.

TODAS LAS ENTRADAS



MUESTRAS



ENTRENOPRUEBA

La brecha de generalización

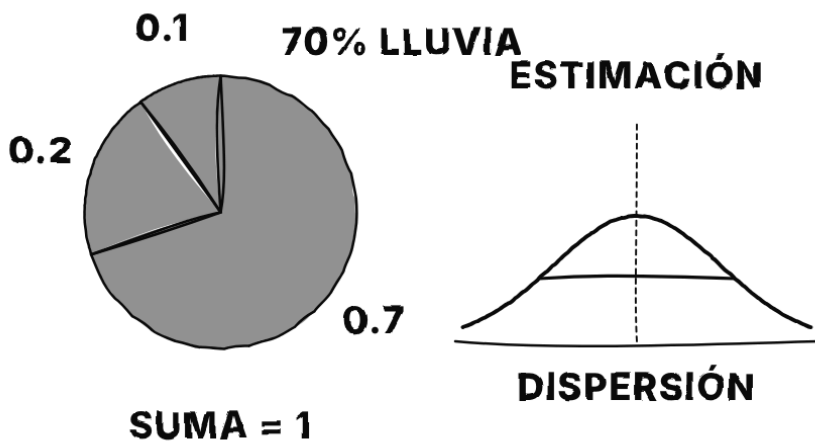
La diferencia entre los dos se llama brecha de generalización. Un modelo que memoriza, como la tabla de consulta, tiene una brecha enorme: sobreajusta, encajando con los datos de entrenamiento concretos en lugar de con el patrón que hay detrás. La prueba práctica es guardar un conjunto de prueba aparte, ejemplos nunca usados durante el entrenamiento, y comparar la pérdida sobre los datos de entrenamiento con la pérdida sobre el conjunto de prueba. Una gran diferencia es señal de sobreajuste. Por eso una aplicación de fotos se juzga con fotos que nunca se le mostraron durante el entrenamiento: solo esas revelan si aprendió lo que es un perro o simplemente memorizó ciertas imágenes.

Pruébalo

Pida a un amigo que memorice las respuestas a diez preguntas de trivial y después pregúntele esas mismas diez: acertará todas. Ahora hágale diez preguntas nuevas sobre los mismos temas. Su segunda nota, no la primera, le dice cuánto ha aprendido de verdad, y la caída entre las dos notas es la propia brecha de generalización de su amigo.

¿Cómo debería elegirse una función de pérdida? El libro sugiere un repaso rápido de su apéndice de probabilidad primero, y después explica que la pérdida lleva nuestro conocimiento previo de la tarea y afecta mucho al rendimiento. A veces el sentido común basta para diseñar una, como con la pérdida de bisagra. Pero hay una vía más fundamentada: describir todo el proceso de entrenamiento en el lenguaje de la probabilidad, un punto de vista que muchos otros libros prefieren.

La idea de partida es que los ejemplos vienen de una fuente desconocida de pares entrada-salida, que se puede dividir en dos piezas: lo probable que es cada entrada, y lo probable que es cada salida una vez conocida la entrada. Los métodos que modelan lo probables que son las entradas se llaman generativos, los que modelan solo la salida dada la entrada se llaman discriminativos, y aparte de los modelos de lenguaje este libro se centra en el segundo tipo. Predecir una única respuesta tiene sentido cuando, para cada entrada, casi toda la probabilidad se concentra en un valor.



Respuestas como distribuciones

El libro relaja eso: en lugar de una respuesta, las salidas del modelo fijan los mandos de toda una distribución de probabilidad sobre las respuestas posibles. Para elegir entre tres clases, el modelo da tres probabilidades que suman uno, un gráfico circular sobre las clases. Para predecir un solo número, el modelo puede dar dos salidas: una mejor estimación y una medida de dispersión, la segunda elevada al cuadrado para que nunca pueda ser negativa. Este planteamiento incluye todo lo dicho hasta ahora, y a menudo es más fácil elegir una distribución sensata que inventar una pérdida a mano. También expresa la incertidumbre directamente, mediante la dispersión. Una aplicación del tiempo que dice «70 por ciento de probabilidad de lluvia» en lugar de simplemente «lluvia» le está dando exactamente este tipo de respuesta.

Para recordar

Una tabla de consulta puede alcanzar la pérdida media mínima en los datos de entrenamiento y ser inútil, lo que muestra que memorizar no es aprender. Lo que importa es el riesgo esperado, la pérdida media sobre todos los datos futuros posibles; no se puede calcular, y el riesgo empírico solo lo estima. La diferencia entre ellos es la brecha de generalización; una brecha grande significa sobreajuste, y un conjunto de prueba aparte la mide. En la visión probabilística, el modelo da como salida los ajustes de una distribución sobre las respuestas, como probabilidades de clase o una estimación con su dispersión, lo que también expresa la incertidumbre.

Preguntas

1. Una tabla de consulta alcanza la pérdida media más baja posible en los datos de entrenamiento. ¿Por qué sigue siendo inútil?
 - a. Porque es demasiado lenta de ejecutar
 - b. Porque solo memoriza los pares de entrenamiento y da un valor fijo por defecto para cualquier cosa nueva
 - c. Porque su pérdida no es diferenciable
 - d. Porque necesita un conjunto de prueba para funcionar
2. ¿Por qué no podemos simplemente entrenar un modelo minimizando directamente el riesgo esperado?

- a. Porque el riesgo esperado siempre es cero
 - b. Porque haría falta listar y etiquetar todas las entradas posibles, lo cual es imposible
 - c. Porque ignora la función de pérdida
 - d. Porque solo funciona para correos electrónicos
3. Un modelo tiene una pérdida muy baja en sus datos de entrenamiento y una pérdida mucho más alta en un conjunto de prueba aparte. ¿Qué indica esto?
- a. El modelo generaliza perfectamente
 - b. El conjunto de prueba debe contener errores
 - c. El modelo está sobreajustando, con una brecha de generalización grande
 - d. La función de pérdida no es diferenciable
4. En la visión probabilística, ¿qué da como salida un modelo para predecir un solo número en lugar de una respuesta?
- a. Una mejor estimación junto con una medida de dispersión
 - b. Tres probabilidades que suman uno
 - c. Solo el valor de la pérdida
 - d. Una tabla de consulta de respuestas pasadas

Respuestas

1. **b)** La tabla devuelve respuestas guardadas para las entradas de entrenamiento y un valor por defecto para el resto, así que nunca aprendió el patrón: memorizar no es aprender.
2. **b)** El riesgo esperado promedia sobre todos los pares entrada-salida posibles, que nadie puede enumerar; el riesgo empírico solo lo estima.

3. c) Una gran diferencia entre la pérdida de entrenamiento y la de prueba es la señal práctica de una brecha de generalización grande, es decir, de sobreajuste.

4. a) Para un solo número el modelo da una mejor estimación y una dispersión, mantenida positiva al elevarla al cuadrado, lo que también expresa su incertidumbre.

Ejercicios

1. Entrena un filtro de spam e informa de su pérdida media sobre los correos con los que se entrenó. Explique por qué ese número por sí solo no puede decirle si el filtro funciona, y qué debería hacer en su lugar.

Pista: Piense en el estudiante y el examen del año pasado.

Respuesta: La pérdida de entrenamiento se puede hacer diminuta incluso por pura memorización, así que dice poco sobre correos nuevos. Guarde un conjunto de prueba aparte con correos nunca usados en el entrenamiento y mida la pérdida ahí; la diferencia con la pérdida de entrenamiento estima la brecha de generalización y revela el sobreajuste.

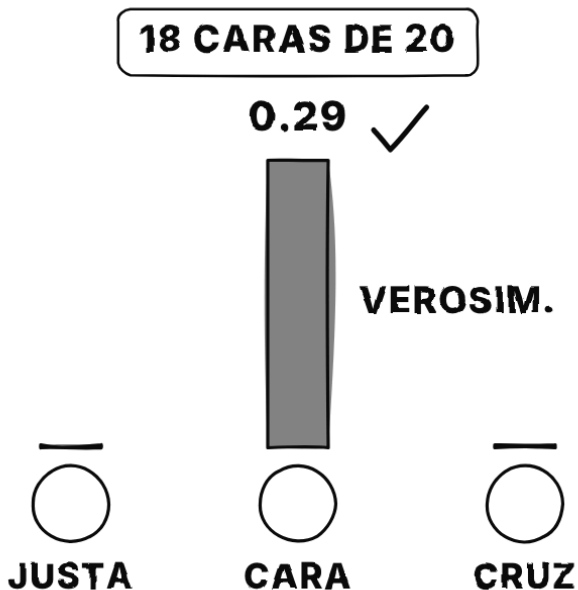
2. Explique con sus propias palabras por qué la visión probabilística puede ser más útil que un modelo que da una única respuesta.

Pista: Piense en la aplicación del tiempo y en elegir una pérdida.

Respuesta: Dar una distribución sobre las respuestas, como probabilidades de clase o una estimación con su dispersión, dice lo seguro que está el modelo, como una probabilidad de lluvia en lugar de un pronóstico tajante. También cubre el planteamiento anterior como caso especial y hace más fácil diseñar el entrenamiento, porque elegir una distribución sensata suele ser más simple que inventar una pérdida a mano.

El modelo más creíble

Imagine que tres amigos afirman cada uno haber lanzado una moneda veinte veces, y uno de ellos le entrega una hoja con dieciocho caras. Un amigo dice que usó una moneda corriente, otro una moneda cargada hacia cara, el tercero una moneda cargada hacia cruz. Sin ninguna fórmula ya sabe qué historia creer: la moneda cargada hacia cara hace esa hoja mucho más probable que las otras. Elegir la explicación que hace más probables los datos observados es la idea detrás de la máxima verosimilitud, y es como el libro entrena los modelos probabilísticos.



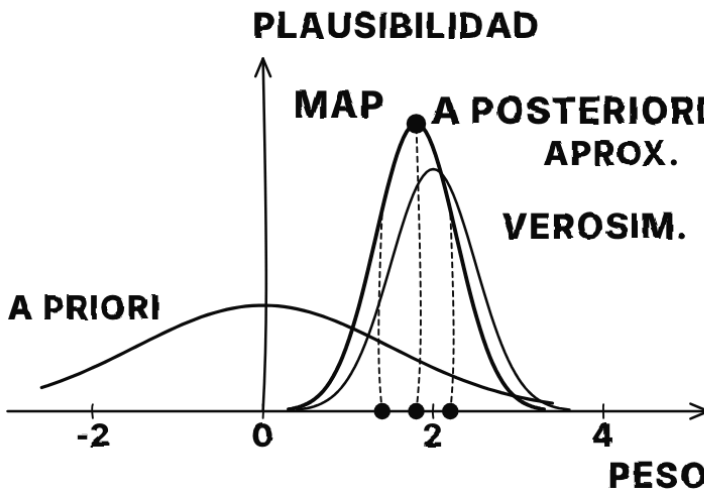
¿Qué moneda explica 18 caras?

Recuerde que los ejemplos de un conjunto de datos se suponen extraídos de forma independiente de la misma fuente desconocida. Así que un modelo que da probabilidades a las respuestas también da una probabilidad al conjunto de datos entero: las probabilidades de todos los ejemplos, multiplicadas entre sí. Ese número se llama la verosimilitud del conjunto de datos. Un modelo elegido al azar reparte sus probabilidades casi a ciegas, así que encuentra muy improbable nuestro conjunto de datos concreto. La elección sensata es el modelo bajo el cual nuestro conjunto de datos es lo más probable posible: la solución de máxima verosimilitud.

Sigue siendo un problema de optimización, pero con una diferencia. Antes, la función de pérdida era una decisión de diseño; aquí, una vez elegidas las distribuciones de probabilidad, el objetivo se sigue directamente de las leyes de la probabilidad. Y aun así las dos visiones se encuentran. Tomar logaritmos convierte el largo producto en una suma, y cambiar el signo convierte maximizar en minimizar, así que la máxima verosimilitud se convierte en minimizar una media de menos logaritmos de probabilidades, una especie de pseudopérdida. El libro añade que toda función de pérdida usada en la práctica se puede obtener así, para una elección adecuada de distribución, e insta al lector a tener presentes ambas visiones. El entrenamiento de un asistente de chat, que aprende a dar alta probabilidad a la siguiente palabra real de su texto de entrenamiento, es máxima verosimilitud a una escala gigantesca.

El libro da entonces un paso más, hacia el aprendizaje bayesiano y las redes neuronales bayesianas, dando solo la idea principal. Incluso un modelo probabilístico, una vez entrenado, sigue siendo una sola función, pero muchos ajustes distintos de un modelo podrían explicar los datos igual de bien, y conservarlos todos podría hacer las

predicciones más fieles. La receta es de nuevo dejar que la probabilidad nos guíe. Primero, enunciar una distribución a priori: lo plausible que parece cada función posible antes de ver ningún dato. En la práctica eso significa una distribución a priori sobre los pesos, y una elección común favorece los pesos más pequeños, porque esas funciones tienden a ser más estables.



A priori por verosimilitud

Cuando llegan los datos, el teorema de Bayes combina la a priori con la verosimilitud para dar la a posteriori, la **plausibilidad actualizada** de cada función tras ver los datos; un término normalizador llamado **evidencia** la convierte en una distribución propiamente dicha. Para predecir, se promedian las predicciones de todas las funciones, cada una ponderada por su **plausibilidad a posteriori**, y en la práctica esto

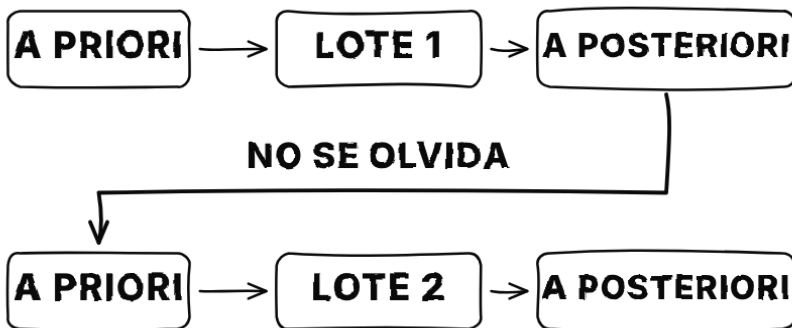
se aproxima muestreando un puñado de funciones y promediándolas. El problema es que la *a posteriori* casi nunca se puede calcular con exactitud, así que hacen falta aproximaciones, con métodos como las cadenas de Markov Monte Carlo o la inferencia variacional. El capítulo nueve mostrará un ejemplo práctico sencillo, una técnica llamada dropout Monte Carlo.

Pruébelo

Llene un tarro con canicas de dos colores en una proporción que no revela, y pida a alguien que adivine la proporción antes de ver ninguna canica: esa suposición es su *a priori*. Ahora déjele sacar una canica a la vez y actualizar su suposición tras cada extracción. Fíjese en que cada suposición actualizada se convierte en el punto de partida para la siguiente extracción: la conclusión de ayer es la *a priori* de hoy.

Dos observaciones cierran la sección. Primera: si solo queremos la única función más plausible, podemos ignorar la evidencia y maximizar juntas la verosimilitud y la *a priori*, lo que se llama la solución máxima *a posteriori*. Si la *a priori* trata todas las funciones por igual, vuelve a ser simplemente máxima verosimilitud. En otro caso la *a priori* penaliza las funciones que se alejan demasiado de lo que esperábamos, y esa penalización, llamada regularizador, es un arma útil contra el sobreajuste. El libro señala que la diferencia refleja a grandes rasgos dos formas de leer la probabilidad: como la frecuencia de los sucesos, tratando los parámetros como fijos y los datos como aleatorios, o como una medida de incertidumbre, tratando los datos como fijos y los parámetros como inciertos.

Segunda: la visión bayesiana hace natural aprender de datos nuevos a medida que llegan. La a posteriori calculada con el primer lote de datos se convierte simplemente en la a priori para el siguiente lote, que entonces la actualiza de nuevo. Esto puede aliviar un problema notorio de los modelos entrenados de forma continua, llamado olvido catastrófico, en el que aprender información nueva borra la antigua. Una aplicación que sigue aprendiendo de su comportamiento se enfrenta exactamente a este riesgo de olvidar lo que aprendió antes, y esta forma de actualizar creencias es una vía para suavizarlo.



La a posteriori de hoy, la a priori de mañana

Para recordar

La verosimilitud es la probabilidad que un modelo da al conjunto de datos entero, y la máxima verosimilitud elige el modelo que hace más probables los datos observados. En forma logarítmica se convierte en minimizar una suma de menos logaritmos de probabilidades, y toda función de pérdida práctica se puede derivar así. El aprendizaje bayesiano conserva toda una distribución sobre las funciones: una a priori antes de los datos, actualizada por el teorema de Bayes en una a posteriori, con predicciones promediadas sobre las funciones plausibles, aunque normalmente solo de forma aproximada. La única función más plausible, la solución máxima a posteriori, añade la a priori como regularizador contra el sobreajuste, y la a posteriori puede servir como a priori para datos nuevos para combatir el olvido catastrófico.

Preguntas

1. ¿Por qué un modelo elegido al azar da una verosimilitud muy baja a nuestro conjunto de datos?
 - a. Porque reparte sus probabilidades casi a ciegas, así que nuestros ejemplos concretos reciben poca probabilidad
 - b. Porque las verosimilitudes siempre son cero antes del entrenamiento
 - c. Porque los modelos aleatorios no pueden multiplicar probabilidades

- d. Porque el conjunto de datos no se recogió de forma independiente
2. ¿Cómo se conecta la máxima verosimilitud con las funciones de pérdida vistas antes?
- a. Sustituye el descenso de gradiente por un recuento
 - b. Tomar logaritmos y cambiar el signo la convierte en minimizar una suma de menos logaritmos de probabilidades, una pseudopérdida
 - c. Solo funciona con la pérdida de bisagra
 - d. No tiene ninguna conexión con las pérdidas
3. En la solución máxima a posteriori, ¿qué pasa si la a priori trata todas las funciones como igual de plausibles?
- a. Se convierte en la solución de máxima verosimilitud simple
 - b. Siempre elige los pesos más pequeños
 - c. No se puede calcular
 - d. Ignora los datos por completo
4. ¿Cómo ayuda la visión bayesiana a que un modelo siga aprendiendo de datos nuevos sin olvidar los antiguos?
- a. Borra los datos antiguos después de cada lote
 - b. La a posteriori de los datos anteriores se convierte en la a priori para los datos nuevos
 - c. Entrena un modelo completamente nuevo para cada lote
 - d. Pone todos los pesos a cero antes de cada actualización

Respuestas

1. **a)** La verosimilitud multiplica las probabilidades que el modelo da a todos los ejemplos, y un modelo aleatorio da poca probabilidad a nuestros ejemplos concretos.
2. **b)** El logaritmo convierte el producto en una suma y el cambio de signo convierte maximizar en minimizar, así que se vuelve una minimización de pérdida ordinaria.
3. **a)** Con una a priori uniforme el término a priori es constante, así que solo importa la verosimilitud y la solución es la máxima verosimilitud.
4. **b)** La conclusión de ayer se convierte en la a priori de hoy, así que la actualización se construye sobre lo aprendido, aliviando el olvido catastrófico.

Ejercicios

1. Explique con sus propias palabras los papeles de la a priori, la verosimilitud y la a posteriori en el aprendizaje bayesiano, usando el tarro de canicas.

Pista: ¿Qué creencia viene antes de sacar, cuál viene de las extracciones y cuál después?

Respuesta: La a priori es la suposición sobre la proporción de colores antes de sacar ninguna canica. La verosimilitud dice lo probables que son las canicas sacadas bajo cada proporción posible. La a posteriori combina las dos en una creencia actualizada tras las extracciones, y puede servir como a priori para las siguientes extracciones.

2. ¿Por qué el libro llama regularizador al término a priori de la solución máxima a posteriori, y por qué eso ayuda contra el sobreajuste?

Pista: Piense en qué hace la a priori con las funciones que se alejan de lo que esperábamos.

Respuesta: La *a priori* añade una penalización a las funciones que se alejan mucho de lo que esperábamos, por ejemplo funciones con pesos muy grandes. Esto empuja la solución hacia funciones más simples y estables en lugar de hacia las que encajan con cada rareza de los datos de entrenamiento, lo que reduce el sobreajuste.

3. ¿Por qué el aprendizaje bayesiano completo es difícil de usar con exactitud en la práctica, y qué se hace en su lugar?

Pista: Piense en calcular la *a posteriori*.

Respuesta: La *a posteriori* sobre todas las funciones casi nunca se puede calcular con exactitud, salvo para elecciones especiales de *a priori* y verosimilitud. Así que se usan aproximaciones, como las cadenas de Markov Monte Carlo o la inferencia variacional, y las predicciones se promedian sobre un puñado de funciones muestreadas.