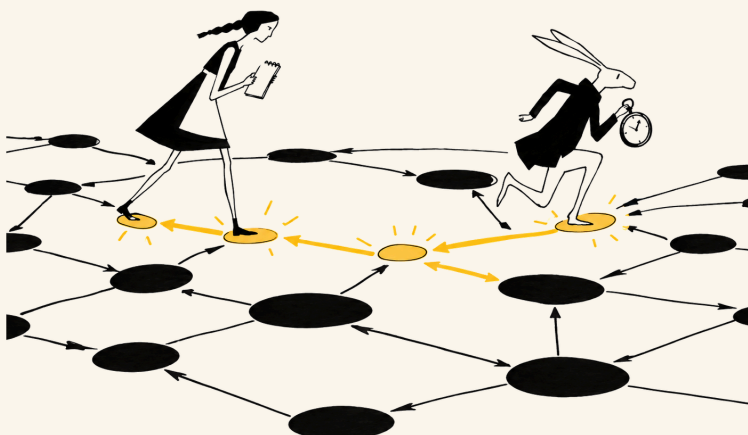


Las aventuras de Alicia en el País de las Maravillas Diferenciables



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

Una cadena de teclas

Una sola línea larga de números

Tres formas de hallar una pendiente

Recorrer la cadena hacia atrás

No construya nunca la tabla

Notas dejadas por el camino

Un cuarto, diez veces seguidas

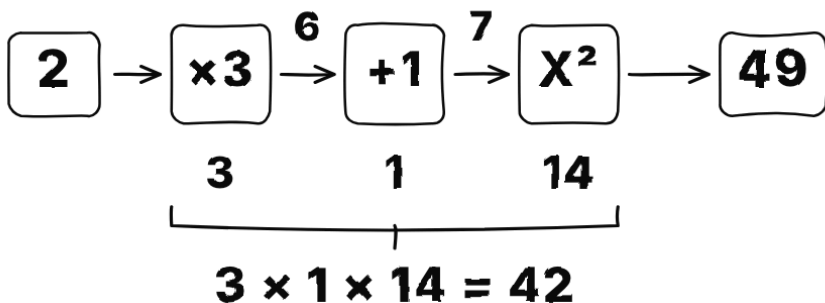
Una regla en la esquina

Una cadena de teclas

Tome una calculadora y pulse una cadena corta de teclas: empiece con dos, multiplique por tres, sume uno y eleve el resultado al cuadrado. Obtiene cuarenta y nueve. Ahora empiece de nuevo desde dos coma cero uno, un empujón de una centésima, y pulse las mismas teclas. Quizá espere que la respuesta se mueva también una centésima, pero sale cerca de cuarenta y nueve coma cuarenta y dos, así que se movió cuarenta y dos veces más que su empujón. ¿De dónde sale el cuarenta y dos?

2.01

49.42

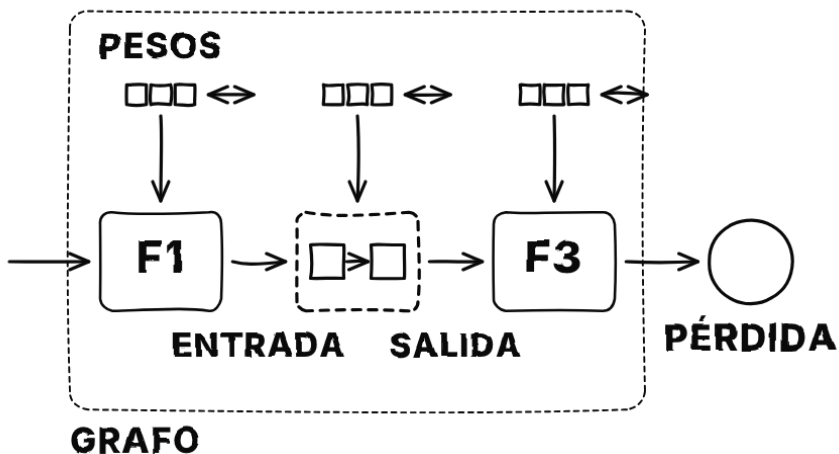


REGLA CADENA

Una cadena de teclas de calculadora

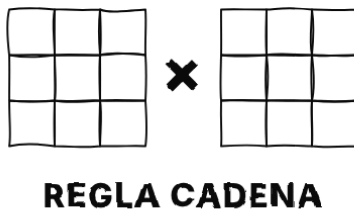
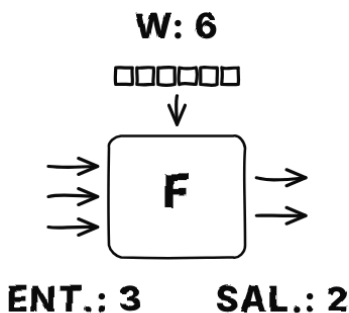
Mire cada tecla por separado. Multiplicar por tres hace cualquier cambio pequeño tres veces mayor. Sumar uno deja pasar el cambio intacto, así que su tasa es uno. Elevar al cuadrado, en el valor siete, hace un cambio pequeño catorce veces mayor, el doble del valor sobre el que actúa. Multiplique las tres tasas locales, tres por uno por catorce, y obtiene cuarenta y dos: la tasa de toda la cadena es el producto de las tasas de sus eslabones. Esa es la regla de la cadena, y este capítulo trata de hacer exactamente esto, y de prisa, para programas con millones de teclas.

Cuando se entrena un asistente de chat o la aplicación de fotos de su teléfono, el programa es una larga cadena de pasos así, y al final hay un único número, la pérdida, que puntúa lo mal que lo hizo el programa con un lote de ejemplos. El entrenamiento necesita saber cómo se mueve esa única puntuación cuando se empuja cada uno de los números ajustables del programa, sus pesos. Calcular todas esas tasas automáticamente a partir del propio programa se llama diferenciación automática. El libro imagina el programa como un grafo computacional: el conjunto de operaciones pequeñas e indivisibles que el programa realiza de verdad al ejecutarse, a las que llama primitivas. Para simplificar, el capítulo trata los pasos como una cadena recta, uno tras otro, pero todo se traslada a programas que se ramifican y se vuelven a unir.



Un programa como cadena de primitivas

Cada primitiva recibe una lista de números, la entrada, usa su propia lista de números ajustables, sus pesos, y entrega una lista nueva, la salida. Una primitiva puede ser tan pequeña como una multiplicación, tan grande como una capa entera de la red, o incluso un bloque hecho de muchas capas; construir operaciones grandes a partir de pequeñas es lo que la programación hace siempre. El libro supone una sola cosa de cada primitiva: que conocemos sus tasas locales, reunidas en dos tablas llamadas jacobianos. El jacobiano de entrada dice cómo se mueve cada número de salida cuando se empuja cada número de entrada, así que tiene una fila por salida y una columna por entrada. El jacobiano de los pesos dice lo mismo para los pesos, con una fila por salida y una columna por peso.



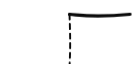
Las dos tablas de una primitiva

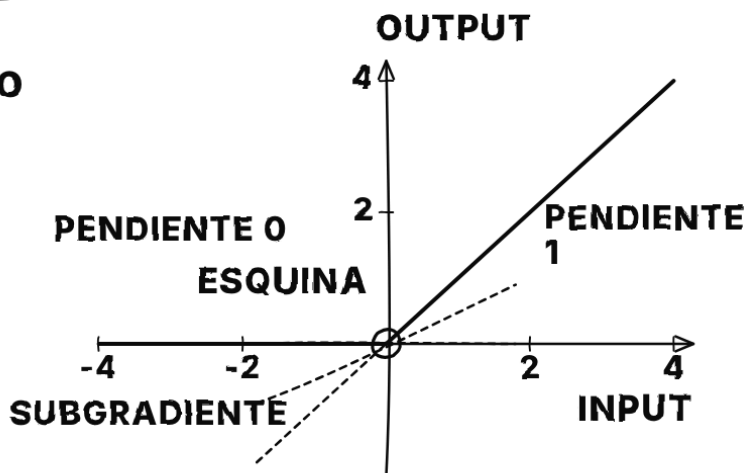
Imagine una primitiva que recibe tres números, tiene seis pesos y entrega dos números. Su jacobiano de entrada es una tabla de dos filas y tres columnas, seis tasas en total; su jacobiano de los pesos tiene dos filas y seis columnas, doce tasas. Ahora imagine una entrada de mil números y una salida de mil: el jacobiano de entrada por sí solo contiene un millón de tasas, y eso es una sola capa de una red modesta.

Al principio, toda la tarea parece trivial. La regla de la cadena, como vio con la calculadora, dice que la tasa de una cadena es simplemente el producto de las tablas de sus eslabones, multiplicadas una tras otra. Lo difícil, y el tema del capítulo, es hacer estas multiplicaciones de forma eficiente. La respuesta a la que llega el capítulo se llama

diferenciación automática en modo inverso, o retropropagación en el lenguaje de las redes neuronales, y es la piedra angular de cómo se entrenan las redes. Entenderla explica también cómo están contruidos los programas con los que se entrenan las redes, como PyTorch, JAX o TensorFlow, y por qué se diferencian.

Queda una duda: ¿y si un paso no tiene tasa en algún punto? La ReLU, un paso habitual que conserva los números positivos y convierte en cero los negativos, tiene una esquina aguda en cero. Los pasos así se tratan con una noción de pendiente más débil, el subgradiente, que llega al final del capítulo. Los pasos que saltan, como sacar una muestra aleatoria o cortar en un umbral, pueden incorporarse sustituyendo su tasa por un sustituto suave; el autor lo deja para el próximo volumen.


SALTO



La ReLU y su esquina

Para recordar

La diferenciación automática calcula, directamente a partir de un programa, cómo cambia una única puntuación final cuando se empuja cada número ajustable del programa. El programa se ve como una cadena de primitivas, cada una de las cuales recibe una entrada y sus propios pesos y entrega una salida. Para cada primitiva solo necesitamos sus dos tablas de tasas locales: el jacobiano de entrada, salidas frente a entradas, y el jacobiano de los pesos, salidas frente a pesos. La regla de la cadena multiplica estas tablas, igual que tres por uno por catorce dio cuarenta y dos en la calculadora; el verdadero reto es hacerlo de forma eficiente, y la retropropagación es la respuesta.

Preguntas

1. En la calculadora, la respuesta final se movió cuarenta y dos veces más que el empujón. ¿Por qué cuarenta y dos?
 - a. Porque cuarenta y dos es la suma de las tasas locales tres, uno y catorce, más el valor inicial
 - b. Porque las tasas locales de las tres teclas, tres, uno y catorce, multiplicadas dan cuarenta y dos
 - c. Porque elevar al cuadrado siempre multiplica un cambio por la respuesta final dividida por el valor inicial
 - d. Porque un empujón de una centésima siempre se amplifica por el número de teclas pulsadas

2. Una primitiva entrega dos números y tiene seis pesos. ¿Qué forma tiene su jacobiano de los pesos?
 - a. Dos filas y seis columnas, una fila por salida y una columna por peso
 - b. Seis filas y dos columnas, una fila por peso y una columna por salida
 - c. Dos filas y dos columnas, porque solo cuentan las salidas
 - d. Seis filas y seis columnas, porque cada peso afecta a cada peso
3. Si la regla de la cadena solo multiplica tablas, ¿por qué dice el capítulo que la tarea es difícil?
 - a. Porque la regla de la cadena solo funciona para programas que se ramifican y se vuelven a unir
 - b. Porque las tasas locales de la mayoría de las primitivas son desconocidas y hay que adivinarlas
 - c. Porque las tablas pueden ser enormes, así que hacer las multiplicaciones de forma eficiente es el verdadero reto
 - d. Porque la puntuación final es una lista de números y no un único número
4. ¿Por qué necesita la ReLU un trato especial en este esquema?
 - a. Porque tiene pesos que no se pueden empujar
 - b. Porque salta al azar, como al sacar una muestra
 - c. Porque cambia el tamaño de la lista de números que recibe
 - d. Porque tiene una esquina aguda en cero, donde no tiene una tasa ordinaria

Respuestas

1. **b)** Cada tecla cambia un empujón pequeño según su propia tasa local, y la tasa de toda la cadena es el producto de las tasas de sus eslabones: tres por uno por catorce.
2. **a)** El jacobiano de los pesos dice cómo se mueve cada salida cuando se empuja cada peso, así que tiene una fila por salida y una columna por peso: dos por seis, doce tasas.
3. **c)** Una sola capa con mil entradas y mil salidas ya tiene un millón de tasas en su jacobiano de entrada, así que lo difícil es hacer las multiplicaciones de forma eficiente, y eso lo resuelve la retropropagación.
4. **d)** La ReLU conserva los números positivos y convierte en cero los negativos, lo que deja una esquina aguda en cero; allí se usa una noción de pendiente más débil, el subgradiente.

Ejemplos resueltos

El tamaño de las dos tablas Una primitiva recibe cuatro números, tiene diez pesos y entrega tres números. ¿Cuántas tasas hay en su jacobiano de entrada y cuántas en su jacobiano de los pesos?

El jacobiano de entrada tiene una fila por salida y una columna por entrada, así que tiene tres filas y cuatro columnas.

Tres por cuatro da doce tasas.

El jacobiano de los pesos tiene una fila por salida y una columna por peso, así que tres filas y diez columnas.

Tres por diez da treinta tasas.

Respuesta: Doce tasas en el jacobiano de entrada y treinta en el jacobiano de los pesos.

Una cadena de tres teclas Empiece en uno, multiplique por cinco, sume dos y eleve al cuadrado. ¿Cuántas veces se mueve la respuesta final en comparación con un empujón pequeño del número inicial?

Multiplicar por cinco hace un cambio pequeño cinco veces mayor: tasa local cinco.

Sumar dos deja pasar el cambio intacto: tasa local uno.

El valor que llega al cuadrado es siete, y elevar al cuadrado hace un cambio pequeño el doble del valor mayor: tasa local catorce.

La tasa de la cadena es el producto de las tasas locales: cinco por uno por catorce.

Respuesta: Setenta veces: un empujón de una centésima mueve la respuesta unas siete décimas, de cuarenta y nueve a unos cuarenta y nueve coma siete.

Ejercicios

1. Una capa recibe cien números, entrega diez y tiene mil diez pesos. ¿Cuántas tasas hay en su jacobiano de entrada y en su jacobiano de los pesos?

Pista: Las filas son siempre las salidas; las columnas son las entradas en una tabla y los pesos en la otra.

Respuesta: Mil y diez mil cien: diez filas por cien columnas en el jacobiano de entrada, diez filas por mil diez columnas en el jacobiano de los pesos.

2. Una cadena tiene dos teclas: multiplicar por dos y después multiplicar por cuatro. Si empuja el número inicial una décima, ¿cuánto se mueve la respuesta final?

Pista: Halle primero la tasa local de cada tecla y luego combínelas como dice la regla de la cadena.

Respuesta: Ocho décimas: las tasas locales dos y cuatro multiplicadas dan ocho, y ocho por una décima son ocho décimas.

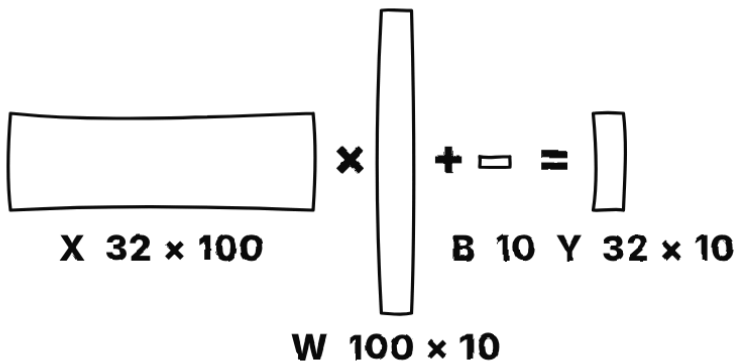
3. Explique con sus propias palabras por qué al capítulo le basta conocer dos tablas de tasas locales de cada primitiva, y no las tasas de todo el programa.

Pista: Piense en lo que hace la regla de la cadena con las tasas de los eslabones.

Respuesta: La regla de la cadena construye la tasa de toda la cadena multiplicando las tasas de sus eslabones. Así que, si se conocen el jacobiano de entrada y el jacobiano de los pesos de cada primitiva, de ahí salen las tasas de la puntuación final respecto de cada peso; el trabajo que queda es hacer esas multiplicaciones de forma eficiente.

Una sola línea larga de números

Tome una hoja de papel cuadriculado y llene un bloque de números, treinta y dos filas de cien números cada una. Ahora cópielos en una sola tira larga, fila tras fila: la primera fila, luego la segunda justo después, y así sucesivamente. No se ha perdido nada: la tira contiene los mismos tres mil doscientos números, y quien conozca la regla puede volver a doblarla en el bloque. Este pequeño truco, desenrollar una tabla en una sola línea, es lo que permite al capítulo hablar de cada capa de una red con las mismas tablas simples de tasas.



Del bloque a la tira, y una capa

Veamos por qué hace falta. Cuando se entrena la aplicación de fotos de su teléfono, las imágenes no pasan de una en una; llegan en un minilote, digamos treinta y dos a la vez. Una capa totalmente conectada básica trabaja entonces sobre una tabla entera: treinta y dos filas, una por imagen, y en cada fila cien características de esa imagen. La capa multiplica esta tabla por una tabla de pesos, de cien filas por diez columnas, y suma diez números extra llamados sesgos, uno por salida. Sale una tabla de treinta y dos filas y diez columnas, diez características nuevas por cada imagen.

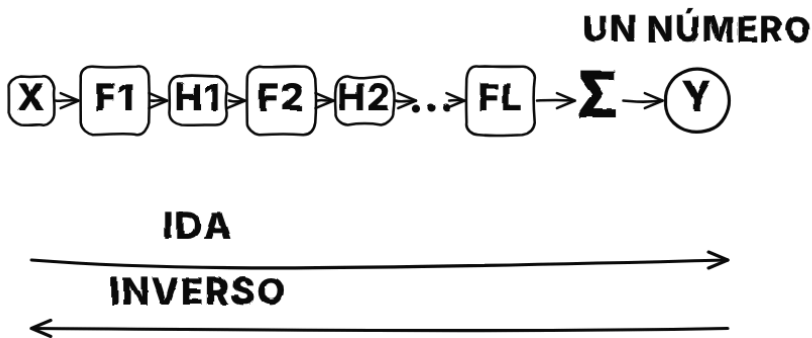
Ahora hágase de nuevo la pregunta de la calculadora: ¿cómo se mueve cada número de salida cuando se empuja cada número de entrada? Una salida se elige con dos posiciones, su imagen y su característica, y lo mismo una entrada, así que la tabla de tasas tiene cuatro direcciones en lugar de dos. El libro dice que estos jacobianos tienen rango cuatro, y son difíciles de imaginar. La salida es la tira. Desenrolle el bloque de entrada en una línea de tres mil doscientos números, desenrolle el bloque de salida en una línea de trescientos veinte, y junte también los pesos y los sesgos en una línea. Entonces el jacobiano de entrada vuelve a ser una tabla simple, una fila por número de salida y una columna por número de entrada: trescientas veinte filas por tres mil doscientas columnas, algo más de un millón de tasas.

De aquí salen dos cosas, y el libro insiste en las dos. Primera: a partir de ahora, cuando habla del tamaño de la entrada, se refiere a la cuenta de todos sus números juntos, lote incluido, así que aquí tres mil doscientos, no cien. Segunda: mire de nuevo ese millón, para una sola capa modesta y un lote pequeño. Podemos saber exactamente cómo calcular una tabla así y aun así no querer construirla en memoria, y esta preocupación marcará todo lo que sigue.

El libro añade una nota sobre cómo se escriben los programas. Su forma de describir una primitiva, una función que recibe su entrada y sus pesos uno al lado del otro, es como funciona una biblioteca funcional como JAX. En PyTorch o TensorFlow una capa es un objeto que guarda dentro sus pesos, y el objeto se llama solo con la entrada. Ese estilo facilita algunas cosas, como esperar a crear los pesos hasta que la primera entrada muestre lo grandes que deben ser, lo que se llama inicialización perezosa; pero añade una capa de envoltorio que hay que atravesar, y los dos estilos acabarán calculando las tasas de maneras distintas.

Ahora, la tarea en sí. El programa es una cadena de pasos escritos uno a uno: la primera primitiva toma la entrada y sus propios pesos y produce un primer resultado intermedio; la segunda toma ese resultado y sus propios pesos y produce el siguiente; y así hasta la última. Una ejecución escrita así se llama traza de evaluación. Los primeros pasos son las capas del modelo, el último paso puntúa cada ejemplo, por ejemplo con la entropía cruzada, y una suma final reúne las puntuaciones del lote en un solo número. Así que el programa termina siempre en un único número, porque el entrenamiento necesita un número que empujar hacia abajo.

TRAZA



Una traza de evaluación que termina en un número

La diferenciación automática, según la definición del libro, consiste en calcular, todo a la vez y de forma eficiente, cómo se mueve ese único número final respecto de cada peso de cada paso, conociendo solo la cadena y las dos tablas locales de cada primitiva. Hay dos grandes familias de métodos para esto, el modo directo y el modo inverso, y solo se diferencian en el orden en que se multiplican las tablas locales. El inverso, la retropropagación, resultará ser aquí mucho más eficiente. El mismo razonamiento se extiende a programas que se ramifican y se vuelven a unir, y a pesos compartidos por varios pasos, que reaparecen mucho más adelante en el libro.

Para recordar

Las tablas de números con varias direcciones, como un minilote de imágenes, se desenrollan en una sola línea larga para que cada jacobiano siga siendo una tabla simple de filas y columnas. Tras desenrollar, el tamaño de una entrada significa todos sus números, lote incluido, e incluso una capa modesta da una tabla de cerca de un millón de tasas que preferiríamos no construir. Un programa se escribe como una cadena de primitivas que termina en una suma de puntuaciones por ejemplo, así que su resultado es siempre un número. La diferenciación automática calcula cómo se mueve ese número con cada peso a la vez, y el modo directo y el inverso solo se diferencian en el orden de las multiplicaciones.

Preguntas

1. ¿Por qué desenrolla el libro una tabla de minilote en una sola línea larga de números?
 - a. Para que cada jacobiano se convierta en una tabla simple de filas y columnas en lugar de un bloque de cuatro direcciones
 - b. Para que la capa necesite menos pesos y funcione más deprisa en el teléfono
 - c. Para que las imágenes del lote puedan procesarse otra vez de una en una
 - d. Para que los sesgos puedan quitarse de la capa sin cambiar su salida

2. Tras desenrollar, ¿qué entiende el libro por tamaño de la entrada de una capa que trabaja con treinta y dos imágenes de cien características cada una?
 - a. Cien, el número de características de una imagen
 - b. Treinta y dos, el número de imágenes del lote
 - c. Tres mil doscientos, todos los números juntos, lote incluido
 - d. Ciento treinta y dos, las características más las imágenes
3. ¿Por qué el programa del enunciado del problema termina siempre en un único número?
 - a. Porque cada primitiva de la cadena entrega exactamente un número
 - b. Porque una suma final reúne las puntuaciones de todos los ejemplos del lote, y el entrenamiento necesita un número que empujar hacia abajo
 - c. Porque la traza de evaluación solo registra la última capa del modelo
 - d. Porque los pesos de todos los pasos son compartidos y suman un solo número
4. ¿Qué separa el modo directo del modo inverso?
 - a. El modo directo funciona para PyTorch y el inverso solo para JAX
 - b. El modo directo necesita los jacobianos de entrada y el inverso solo los de los pesos
 - c. El modo directo trata programas ramificados y el inverso solo cadenas rectas
 - d. Solo el orden en que se multiplican las tablas locales de tasas

Respuestas

1. **a)** Una salida y una entrada se eligen cada una con dos posiciones, así que la tabla de tasas tiene cuatro direcciones; desenrollar ambas en líneas la convierte de nuevo en una tabla simple, una fila por número de salida y una columna por número de entrada.
2. **c)** A partir de ahora, el tamaño de una entrada es la cuenta de todos sus números juntos, lote incluido: treinta y dos por cien.
3. **b)** El último paso puntúa cada ejemplo y una suma final reúne las puntuaciones del lote en un número, que es lo que el entrenamiento empuja hacia abajo.
4. **d)** Los dos calculan las mismas tasas a partir de las mismas tablas locales; solo se diferencian en el orden de las multiplicaciones, y el orden inverso resultará mucho más eficiente.

Ejemplos resueltos

Las tablas de una capa Una capa totalmente conectada recibe un minilote de treinta y dos imágenes con cien características cada una y da diez características por imagen, usando una tabla de pesos de cien por diez. Tras desenrollar, ¿qué tamaño tienen su jacobiano de entrada y su jacobiano de los pesos, contando solo los pesos de cien por diez como hacen las formas del libro?

Desenrolle la entrada: treinta y dos por cien dan tres mil doscientos números.

Desenrolle la salida: treinta y dos por diez dan trescientos veinte números.

El jacobiano de entrada tiene una fila por número de salida y una columna por número de entrada: trescientos veinte por tres mil doscientos, que son un millón veinticuatro mil tasas.

Los pesos se desenrollan en cien por diez, mil números, así que el jacobiano de los pesos es de trescientos veinte por mil, trescientas veinte mil tasas.

Respuesta: El jacobiano de entrada contiene un millón veinticuatro mil tasas y el jacobiano de los pesos, trescientas veinte mil.

Ejercicios

1. Una capa trabaja con un minilote de ocho ejemplos de cincuenta características cada uno y da veinte características por ejemplo, usando una tabla de pesos de cincuenta por veinte. Tras desenrollar, ¿cuántas tasas hay en su jacobiano de entrada y cuántas en su jacobiano de los pesos, contando solo esos pesos?

Pista: Cuente primero todos los números de la entrada, de la salida y de los pesos, lote incluido.

Respuesta: Sesenta y cuatro mil y ciento sesenta mil: la salida tiene ocho por veinte, ciento sesenta números; la entrada, ocho por cincuenta, cuatrocientos; y los pesos, mil; ciento sesenta por cuatrocientos son sesenta y cuatro mil, y ciento sesenta por mil son ciento sesenta mil.

2. Explique con sus propias palabras por qué el libro dice que podemos saber calcular un jacobiano y aun así no querer construirlo en memoria.

Pista: Piense en cuántas tasas tiene incluso una sola capa modesta con un lote pequeño.

Respuesta: Porque, tras desenrollar, incluso una capa modesta con un lote pequeño tiene un jacobiano de cerca de un millón de tasas. Calcularlo es posible, pero guardar tablas así para cada capa ocuparía muchísima memoria, así que un buen método debería evitar construir las enteras.

3. Describa la diferencia entre la forma funcional de escribir una capa, como en JAX, y la forma de objetos, como en PyTorch o

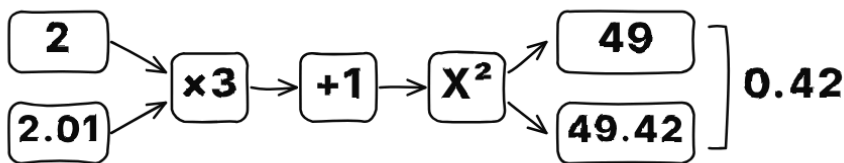
TensorFlow.

Pista: Pregúntese dónde viven los pesos en cada estilo.

Respuesta: En la forma funcional, una primitiva es una función que recibe su entrada y sus pesos uno al lado del otro. En la forma de objetos, la capa guarda dentro sus pesos y se llama solo con la entrada; eso permite la inicialización perezosa, crear los pesos cuando la primera entrada muestra su tamaño, pero añade una capa de envoltorio, y los dos estilos calculan las tasas de maneras distintas.

Tres formas de hallar una pendiente

Vuelva a la cadena de la calculadora del principio: dos, por tres, más uno, al cuadrado, que daba cuarenta y nueve. Suponga que no conociera la regla de la cadena y quisiera la tasa de todos modos. Podría simplemente ejecutar la cadena dos veces, una desde dos y otra desde dos coma cero uno, y dividir la diferencia de las respuestas entre el empujón. Las respuestas son cuarenta y nueve y unos cuarenta y nueve coma cuarenta y dos, así que la tasa sale cerca de cuarenta y dos, un poco por encima porque el empujón no es infinitamente pequeño. Funciona, y es el método más honrado que existe.



$$0,42 \div 0,01 \approx 42$$

✓ SOLO COMPROBAR

□□ □□ □□ □□ □□ ...

2 EJEC. POR PESO

MILES DE MILL.

Empujar y medir

Esto es diferenciación numérica: aplicar directamente la definición de tasa, empujar y medir. El problema es el precio. En una versión sencilla, cada número del que se quiera la tasa cuesta dos ejecuciones del programa entero. Un asistente de chat tiene miles de millones de pesos, así que una sola ronda de entrenamiento necesitaría miles de millones de ejecuciones de toda la red. Por eso, como dice el libro, el método no sirve para entrenar y solo sobrevive como comprobación: se calculan unas pocas tasas empujando y se comparan con lo que dice el método ingenioso.

La segunda forma es la que aprendió en el instituto. Tome una función pequeña: a por el seno de un número, más b por el número por su seno. Un programa de álgebra, como SymPy en Python, puede

calcular exactamente la fórmula de su tasa, como lo haría usted en papel: a por el coseno del número, más b por el número por su coseno, más b por su seno. Esto es diferenciación simbólica. En el número cero, con a y b iguales a uno, la fórmula da uno: el coseno de cero es uno, y los otros dos términos se anulan.

$$f(x) = a \sin x + b x \sin x$$

SYMPY ↓

$$f'(x) = a \cos x + b x \cos x + b \sin x$$

$$X = 0, A = B = 1 \rightarrow 1$$

H = SEN X

GUARDAR 1 VEZ

DIFÍCIL OPTIMIZAR

Una fórmula y sus repeticiones

Pero mire la fórmula: el seno aparece dos veces en la función y el coseno dos veces en la tasa. Un programa sensato calcularía cada uno una vez, guardaría el valor y lo usaría donde hiciera falta. Suena fácil, pero el libro advierte que encontrar la mejor forma posible de organizar un cálculo así, para que nada en absoluto se calcule dos veces, pertenece a una clase de problemas para los que no se conoce ningún método general rápido; se llama acumulación óptima del

jacobiano. La diferenciación automática esquiva esa búsqueda siguiendo la estructura del propio programa, paso a paso, y lo hace mucho mejor que escribir la fórmula completa; de hecho, equivale a una diferenciación simbólica a la que se le permite guardar y reutilizar sus resultados intermedios.

Vamos ahora al primero de los dos métodos automáticos, el modo directo. Parte de la regla de la cadena para dos pasos. Digamos que una primera primitiva convierte la entrada en un resultado intermedio, y una segunda convierte ese resultado en la salida. La tabla de tasas de la salida respecto de la entrada es la tabla del segundo paso multiplicada por la tabla del primero. Los tamaños deben encajar: si la entrada tiene tres números, el resultado intermedio cuatro y la salida dos, la segunda tabla tiene dos filas y cuatro columnas, la primera tiene cuatro filas y tres columnas, y su producto tiene dos filas y tres columnas, exactamente una tasa por cada salida frente a cada entrada. Lea la regla como una receta: una vez que ha ejecutado el primer paso y conoce su tabla, ejecutar el segundo paso le permite actualizar esa tabla multiplicándola por la nueva.

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} & \times & \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} & = & \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \\
 2 \times 4 & & 4 \times 3 & & 2 \times 3
 \end{array}$$



J·T1

J·T2

T3



Σ TASAS

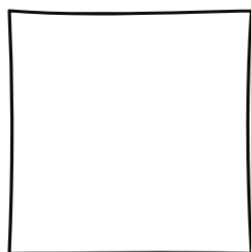
El modo directo lleva tangentes

El modo directo convierte esta receta en un algoritmo. Para cada lista de pesos guarda una tabla llamada tangente: cómo se mueve el resultado intermedio actual respecto de esos pesos. En el primer paso todavía no hay nada guardado, así que la tangente de los primeros pesos es simplemente el jacobiano de los pesos del primer paso. En el segundo paso ocurren dos cosas a la vez: la primera tangente se actualiza, multiplicada por el jacobiano de entrada del segundo paso, y empieza una tangente nueva para los segundos pesos, igual al jacobiano de los pesos del segundo paso. Y así a lo largo de la cadena: en cada paso, cada tangente ya guardada se multiplica por el jacobiano de entrada de ese paso, el mismo para todas, y nace una tangente nueva a partir del jacobiano de los pesos del paso. Al final del todo, el programa suma los números del último resultado, y sumar

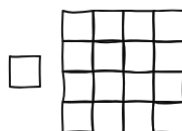
las filas de cada tangente da las tasas de la puntuación final respecto de cada peso.

Luego el libro sopesa el método. En el lado bueno, todo este trabajo puede entretenerse con el programa mientras se ejecuta, paso a paso, así que la memoria necesaria es más o menos la que necesita el propio programa. En el lado malo, el corazón del algoritmo, actualizar una tangente, multiplica dos tablas grandes, y eso es muy caro. Suponga que cada paso recibe y entrega una tabla de minilote de treinta y dos ejemplos con cien características cada uno, y que una tangente pertenece a mil pesos. Desenrollado, el jacobiano de entrada tiene tres mil doscientas filas por tres mil doscientas columnas, y la tangente, tres mil doscientas filas por mil columnas. Su producto cuesta unas tres mil doscientas por tres mil doscientas por mil multiplicaciones: unos diez mil millones, para una actualización de una tangente en un solo paso. El coste crece con el cuadrado del tamaño del lote y con el cuadrado del número de características, así que duplicar ambos lo hace dieciséis veces mayor; para imágenes, con su enorme número de características, pronto se vuelve imposible.

J ENTRADA TANGENTE



×



×2 LOTE
×2 CARACT.

×16

3,200 × 3,200

3,200 × 1,000

≈ 10 000 000 000

✓ **MEMORIA: BAJA**

El precio de actualizar una tangente

Pruébalo

Pruebe el método honrado en una calculadora solo con el cuadrado: eleve al cuadrado tres, y luego tres coma cero cero uno. Las respuestas son nueve y unos nueve coma cero cero seis, así que la tasa se acerca a seis, el doble de tres, como dice la regla escolar. Ahora imagine repetir esas dos ejecuciones para cada uno de los miles de millones de pesos de un asistente de chat, y verá por qué el método honrado se reserva solo para comprobar.

Para recordar

La diferenciación numérica empuja cada número y vuelve a ejecutar el programa, así que cuesta dos ejecuciones por peso y solo sirve para comprobar otros métodos. La diferenciación simbólica escribe la fórmula completa de la tasa, que repite trabajo salvo que se guarden los valores intermedios, y organizar eso a la perfección es un problema difícil; la diferenciación automática sigue en cambio la estructura propia del programa. El modo directo guarda una tangente por cada lista de pesos, multiplica cada tangente guardada por el jacobiano de entrada de cada paso nuevo y añade una tangente nueva a partir del jacobiano de los pesos del paso. Necesita poca memoria, pero cada actualización multiplica dos tablas grandes, un coste que crece con el cuadrado del tamaño del lote y del número de características: diez mil millones de multiplicaciones para una capa modesta.

Preguntas

1. ¿Por qué la diferenciación numérica se reserva solo como comprobación y no se usa para entrenar?
 - a. Porque da una tasa equivocada siempre que la función contiene un seno
 - b. Porque necesita la fórmula completa de la tasa antes de poder empezar

- c. Porque cada peso cuesta dos ejecuciones del programa entero, y las redes tienen miles de millones de pesos
 - d. Porque no puede usarse en una calculadora, solo dentro de un entorno
2. ¿Qué hace derrochadora la fórmula simbólica sin más, y qué hace en su lugar la diferenciación automática?
- a. La fórmula repite términos como el seno y el coseno; la diferenciación automática sigue los pasos propios del programa y reutiliza los resultados intermedios
 - b. La fórmula siempre falla en cero; la diferenciación automática empuja la entrada en su lugar
 - c. La fórmula necesita una calculadora; la diferenciación automática solo necesita papel
 - d. La fórmula tiene demasiado pocos términos; la diferenciación automática añade los que faltan
3. En el modo directo, ¿qué les pasa a las tangentes ya guardadas cuando el programa ejecuta un paso nuevo?
- a. Se tiran y se reconstruyen a partir del jacobiano de los pesos del paso nuevo
 - b. Cada una se multiplica por el jacobiano de entrada del paso nuevo, el mismo para todas, y se empieza una tangente nueva
 - c. Solo se actualiza la tangente más reciente; las anteriores se quedan como estaban
 - d. Se suman todas en una sola tangente para todo el programa
4. Si se duplican a la vez el tamaño del lote y el número de características, ¿cómo cambia el coste de una actualización de

tangente en el modo directo?

- a. Se duplica
- b. Se multiplica por cuatro
- c. Se queda igual, porque el número de pesos no cambia
- d. Se multiplica por dieciséis

Respuestas

1. **c)** Empujar y medir funciona, pero cada número del que se quiere la tasa cuesta dos ejecuciones del programa entero, y un asistente de chat tiene miles de millones de pesos.

2. **a)** El seno aparece dos veces en la función y el coseno dos veces en la tasa; la diferenciación automática sigue la estructura del programa y equivale a una diferenciación simbólica a la que se le permite guardar y reutilizar resultados intermedios.

3. **b)** En cada paso, cada tangente ya guardada se multiplica por el jacobiano de entrada de ese paso, compartido por todas, y nace una tangente nueva a partir del jacobiano de los pesos del paso.

4. **d)** El coste crece con el cuadrado del tamaño del lote y con el cuadrado del número de características, así que duplicar ambos lo multiplica por cuatro y otra vez por cuatro.

Ejemplos resueltos

La tasa de la función con seno La función es $a \sin(bx)$ por el seno de un número, más b por el número por su seno. Su tasa, calculada simbólicamente, es $a \cos(bx)$ por el coseno, más b por el número por el coseno, más b por el seno. ¿Cuál es la tasa en el número cero cuando a vale dos y b vale uno?

El primer término es a por el coseno de cero; el coseno de cero es uno, así que da dos.

El segundo término tiene como factor el propio número, y el número es cero, así que da cero.

El tercer término es b por el seno de cero; el seno de cero es cero, así que da cero.

Sume los tres términos.

Respuesta: Dos.

El precio de una actualización de tangente Un paso recibe y entrega un minilote de treinta y dos ejemplos con cien características cada uno, y una tangente pertenece a mil pesos. ¿Cuántas multiplicaciones cuesta, más o menos, una actualización de esa tangente?

Desenrolladas, la entrada y la salida tienen cada una treinta y dos por cien, tres mil doscientos números.

Así que el jacobiano de entrada es de tres mil doscientos por tres mil doscientos, y la tangente, de tres mil doscientos por mil.

Multiplicar las dos tablas cuesta más o menos el producto de los tres tamaños: tres mil doscientos por tres mil doscientos por mil.

Tres mil doscientos al cuadrado son diez millones doscientos cuarenta mil; por mil dan unos diez mil millones.

Respuesta: Unos diez mil millones de multiplicaciones, para una actualización de una tangente en un solo paso.

Ejercicios

1. Un modelo pequeño tiene mil pesos. ¿Cuántas ejecuciones del programa entero necesita la diferenciación numérica sencilla para obtener todas sus tasas?

Pista: Cada número del que se quiere la tasa cuesta un número fijo de ejecuciones.

Respuesta: Dos mil ejecuciones: dos, con y sin el empujón, por cada uno de los mil pesos.

2. Una primera primitiva convierte cinco números en seis, y una segunda convierte esos seis en dos. ¿Qué formas tienen las dos tablas de la regla de la cadena y su producto?

Pista: Cada tabla tiene una fila por salida de su paso y una columna por entrada de su paso.

Respuesta: La tabla del segundo paso es de dos por seis, la del primero de seis por cinco, y su producto es de dos por cinco: una tasa por cada una de las dos salidas frente a cada una de las cinco entradas.

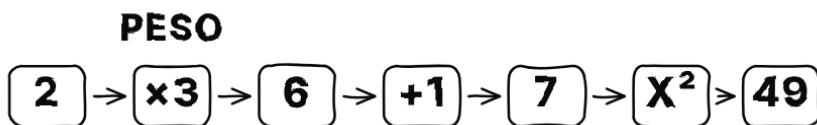
3. En el modo directo, el programa está ejecutando su cuarto paso. ¿Cuántas tangentes guardadas se actualizan en este paso, y cuántas tangentes nuevas se empiezan?

Pista: Cuente las tangentes que han dejado los tres primeros pasos.

Respuesta: Se actualizan tres, una por cada paso anterior, todas multiplicadas por el jacobiano de entrada del cuarto paso, y se empieza una tangente nueva a partir del jacobiano de los pesos del cuarto paso.

Recorrer la cadena hacia atrás

Tome una vez más la cadena de la calculadora: dos, por tres, más uno, al cuadrado, cuarenta y nueve. Esta vez léala desde el final. La última tecla, elevar al cuadrado en el siete, amplifica un cambio pequeño catorce veces, así que la respuesta se mueve catorce veces más que el valor justo antes del cuadrado. Una tecla antes, sumar uno deja pasar los cambios sin alterarlos, así que la respuesta sigue moviéndose catorce veces más que el seis que salió de la multiplicación. Una tecla más atrás, multiplicar por tres, y la respuesta se mueve cuarenta y dos veces más que el dos inicial. Volviendo hacia atrás, aprendió en cada parada cuánto le importa a la respuesta final el número que está allí.



42 ← 14 ← 14

←
ATRÁS

$$14 \times 2 = 28$$

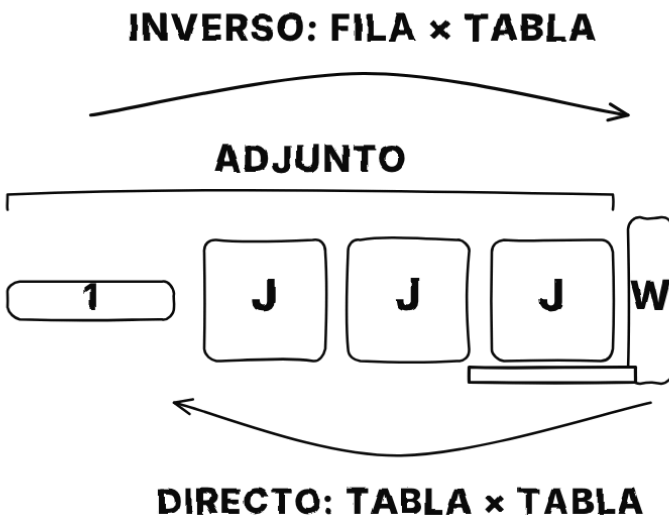
$$3,01 \rightarrow 49,28$$

Leer la cadena desde el final

Ahora imagine que el tres de la primera tecla es un peso ajustable. ¿Cuánto se mueve la respuesta cuando se empuja ese peso? La salida de esa tecla es el peso por dos, así que empujar el peso mueve la salida el doble; y ya sabemos que la respuesta se mueve catorce veces más que esa salida. Catorce por dos: la respuesta se mueve veintiocho veces más que el peso. Puede comprobarlo: con tres coma cero uno en lugar de tres, la cadena da unos cuarenta y nueve coma veintiocho. Ese pequeño camino de vuelta es toda la idea de esta parte.

El libro escribe entera la tasa de la puntuación final respecto de los pesos de un paso, en algún punto del medio de la cadena. Es un producto largo. Primero viene una fila de unos, que representa la suma final: cada número del último resultado cuenta una vez en la

puntuación. Luego vienen los jacobianos de entrada de todos los pasos posteriores, desde el último hacia atrás hasta el paso justo después del nuestro, y por último el jacobiano de los pesos de nuestro propio paso. El libro da un nombre a la primera parte de ese producto, la fila de unos multiplicada por todos esos jacobianos de entrada posteriores: es la tasa de la puntuación final respecto del resultado intermedio de nuestro paso.

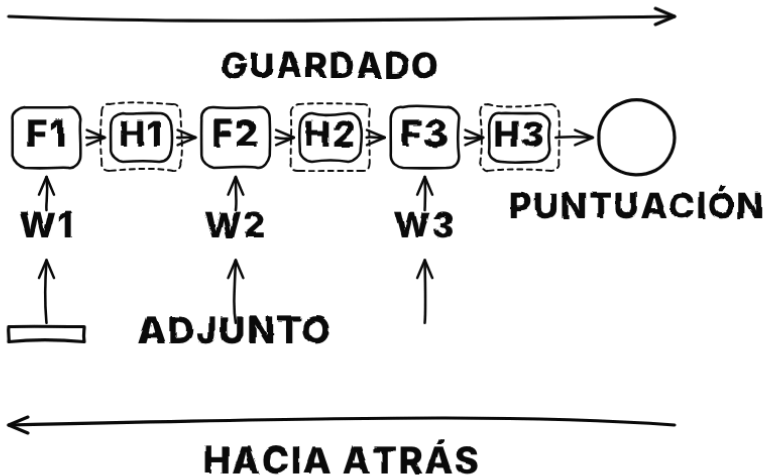


Por qué extremo empezar

En un producto de tablas puede multiplicar en el orden que quiera y obtener el mismo resultado. El modo directo multiplicaba desde el extremo del propio paso hacia fuera, en el orden en que se ejecuta el programa, y pagaba productos grandes de tabla por tabla. Pero mire el otro extremo. Empieza con una sola fila, y una fila por una tabla es

mucho más barato que una tabla por una tabla; lo que sale es otra vez una sola fila. Y la fila de un paso puede obtenerse de la fila del paso siguiente con una multiplicación más por un jacobiano de entrada, volviendo hacia atrás exactamente como hizo en la calculadora.

Eso da la diferenciación automática en modo inverso. Primero ejecute todo el programa, de principio a fin, y guarde cada resultado intermedio. Luego empiece con la fila de unos y vuelva hacia atrás desde el último paso hasta el primero. En cada paso, multiplique la fila actual por el jacobiano de los pesos del paso: esa es la tasa de la puntuación respecto de los pesos de ese paso, y listo. Luego multiplique la fila por el jacobiano de entrada del paso y lleve la fila nueva un paso atrás.

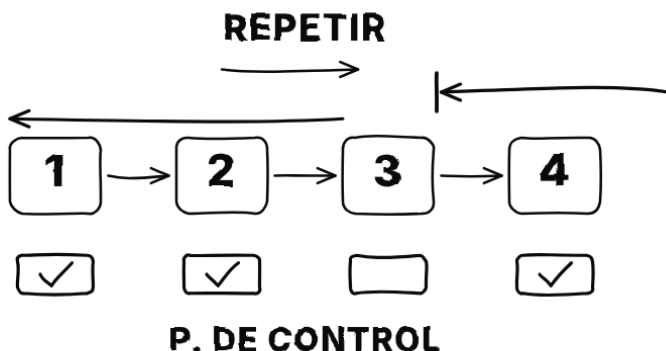


La pasada hacia atrás

Este camino de vuelta es un segundo programa, casi un reflejo del primero, llamado programa inverso o dual. Las filas que lleva se llaman adjuntos: cada una dice cómo se mueve la puntuación final con cada número de un resultado intermedio, mientras que las tangentes del modo directo decían cómo se movía un resultado intermedio con los pesos. En el lenguaje de las redes, ejecutar el programa original es la pasada hacia delante, que no es lo mismo que el modo directo, y el camino de vuelta es la pasada hacia atrás.

La ganancia es grande. Cada paso hacia atrás multiplica una sola fila por una tabla, un coste que crece solo en proporción a los tamaños en juego. Con la capa de antes, treinta y dos ejemplos de cien características y mil pesos, los dos productos de un paso hacia atrás cuestan unos diez millones y unos tres millones de multiplicaciones, más o menos mil veces menos que los diez mil millones de una actualización en modo directo. Esto es lo que permite que la aplicación de fotos de su teléfono, o un asistente de chat, puedan entrenarse siquiera.

PASO ATRÁS $\approx$ 13 MILLONES
DIRECTO $\approx$ 10 000 MILLONES
 $\approx$ 1000 $\times$ MENOS



+25 % TRABAJO

Pasos baratos, memoria cara

El precio es la memoria. Toda la pasada hacia delante debe terminar antes de que empiece la pasada hacia atrás, y cada resultado intermedio debe guardarse hasta que el camino de vuelta llegue a él. Los puntos de control del gradiente cambian parte de esa memoria por trabajo extra. La figura del libro muestra cuatro bloques en fila: durante la pasada hacia delante solo se guardan las salidas del primer, segundo y cuarto bloque, como puntos de control. Al volver, la pasada se atasca en el tercer bloque, cuyos resultados se tiraron; así que repite una breve pasada hacia delante desde el punto de control más cercano, el segundo bloque, reconstruye los resultados que faltan y sigue. Eso cuesta una cuarta parte más de cálculo que una pasada hacia atrás simple, y en general, cuantos menos puntos de control se

guardan, más hay que volver a calcular; el entrenamiento de los modelos muy grandes se apoya en este intercambio cada día.

Para recordar

La tasa de la puntuación final respecto de los pesos de un paso es un producto largo que empieza con una fila de unos y recorre los jacobianos de entrada de los pasos posteriores hasta el jacobiano de los pesos del paso. El modo inverso multiplica desde el extremo de la fila: ejecuta primero todo el programa, guarda cada resultado intermedio y luego vuelve hacia atrás llevando una sola fila, el adjunto. En cada paso, el adjunto por el jacobiano de los pesos da las tasas de ese paso, y el adjunto por el jacobiano de entrada se convierte en el adjunto del paso anterior, igual que catorce por dos dio veintiocho en la calculadora. Los productos de fila por tabla lo hacen unas mil veces más barato que el modo directo para una capa modesta, a costa de memoria, que los puntos de control del gradiente reducen recalculando algunos resultados.

Preguntas

1. ¿Por qué el modo inverso multiplica el producto largo empezando por la fila de unos?
 - a. Porque la fila de unos es la tabla más grande y conviene gastarla primero

- b. Porque una fila por una tabla es mucho más barato que una tabla por una tabla, y el resultado vuelve a ser una sola fila
- c. Porque las tablas solo pueden multiplicarse en el orden en que se ejecutó el programa
- d. Porque la fila de unos elimina la necesidad de los jacobianos de los pesos

2. ¿Qué le dice un adjunto?

- a. Cómo se mueve un resultado intermedio cuando se empujan los pesos de pasos anteriores
- b. Cuántos resultados intermedios hay que guardar durante la pasada hacia delante
- c. Cómo se mueve la puntuación final con cada número de un resultado intermedio
- d. Cuánta memoria necesita un paso durante la pasada hacia atrás

3. ¿Por qué no puede empezar la pasada hacia atrás antes de que termine la pasada hacia delante?

- a. Porque el camino de vuelta empieza en el último paso y necesita los resultados intermedios que la pasada hacia delante produce y guarda
- b. Porque los pesos solo se crean al final de la pasada hacia delante
- c. Porque la fila de unos se calcula sumando todos los pesos
- d. Porque el entorno debe traducir primero el programa a una fórmula

4. ¿Qué ocurre cuando se guardan menos puntos de control?

- a. La pasada hacia atrás se vuelve más barata y necesita menos memoria
- b. Las tasas se vuelven menos exactas, porque algunos resultados se adivinan
- c. Hay que saltarse por completo la pasada hacia delante
- d. Se usa menos memoria, pero hay que volver a calcular más resultados durante la pasada hacia atrás

Respuestas

- 1. **b)** El producto puede hacerse en cualquier orden; empezando por el extremo de la fila, cada multiplicación es una sola fila por una tabla, lo que es barato y vuelve a dar una sola fila.
- 2. **c)** Los adjuntos son las filas que lleva el camino de vuelta, y cada uno da cómo se mueve la puntuación final con los números de un resultado intermedio; las tangentes del modo directo iban en sentido contrario.
- 3. **a)** El camino empieza en el final de la cadena y usa cada resultado intermedio a la vuelta, así que todo el programa debe ejecutarse primero y guardarlos todos.
- 4. **d)** Los puntos de control cambian memoria por trabajo: los resultados que se tiraron se reconstruyen repitiendo parte de la pasada hacia delante, así que menos puntos de control significan más recálculo.

Ejemplos resueltos

El coste de cuatro bloques Una red tiene cuatro bloques. En la pasada hacia delante solo se guardan las salidas del primer, segundo y cuarto bloque. ¿Cuánto cálculo más necesita esto en comparación con una pasada hacia atrás simple, y por qué?

La pasada hacia delante ejecuta los cuatro bloques una vez cada uno.

Al volver, la pasada se atasca en el tercer bloque, cuyos resultados no se guardaron.

Repite el cálculo hacia delante desde el punto de control más cercano, la salida del segundo bloque, lo que significa ejecutar el tercer bloque una vez más.

Así que hacen falta cinco ejecuciones de bloque en lugar de cuatro, una extra por cada cuatro.

Respuesta: Una cuarta parte más de cálculo, uno coma veinticinco veces más.

Un camino de vuelta con un peso Una cadena empieza en uno, multiplica por un peso igual a cinco, suma dos y eleva al cuadrado, y da cuarenta y nueve. Volviendo hacia atrás, ¿cuánto se mueve la respuesta cuando se empuja el peso?

El cuadrado actúa sobre siete y amplifica un cambio pequeño catorce veces: la respuesta se mueve catorce veces más que el siete.

Sumar dos deja pasar los cambios: la respuesta se mueve catorce veces más que el cinco que sale de la multiplicación.

Ese cinco es el peso por el uno inicial, así que empujar el peso lo mueve exactamente lo mismo: tasa uno.

Multiplique la tasa que se lleva por esta tasa local: catorce por uno.

Respuesta: Catorce veces más que el empujón del peso.

Ejercicios

1. Una cadena empieza en tres, multiplica por un peso igual a cuatro, suma uno y eleva al cuadrado. ¿Cuánto se mueve la respuesta en comparación con un empujón pequeño del peso?

Pista: Vuelva hacia atrás desde el cuadrado: halle primero cuánto le importa a la respuesta el número que entra en el cuadrado.

Respuesta: Setenta y ocho veces: el cuadrado actúa sobre trece, así que allí la tasa es veintiséis; sumar uno la deja pasar; la salida de la multiplicación es el peso por tres, así que empujar el peso la mueve el triple; veintiséis por tres son setenta y ocho.

2. Una red tiene ocho bloques, y las salidas de todos los bloques salvo el quinto se guardan como puntos de control. ¿Cuánto cálculo extra necesita la pasada hacia atrás?

Pista: Cuente cuántas ejecuciones de bloque hace la pasada hacia delante y cuántas hay que repetir.

Respuesta: Una octava parte más: la pasada hacia delante ejecuta ocho bloques, y solo hay que repetir el quinto desde el punto de control del cuarto bloque, así que hacen falta nueve ejecuciones en lugar de ocho.

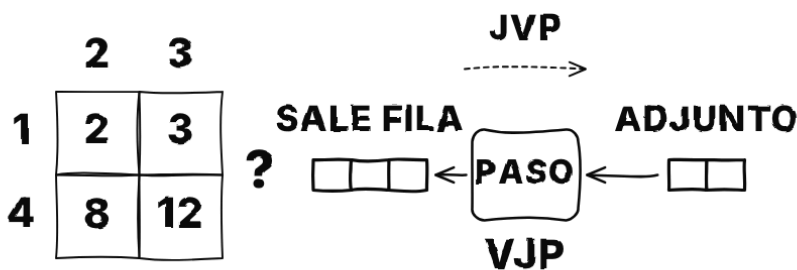
3. Explique con sus propias palabras por qué el modo inverso es tanto más barato que el directo, y qué paga a cambio.

Pista: Compare lo que se multiplica en cada paso en los dos métodos.

Respuesta: El modo directo multiplica tablas grandes por tablas grandes en cada paso, mientras que el modo inverso solo multiplica una fila por una tabla, cuyo coste crece solo en proporción a los tamaños; para una capa modesta eso es unas mil veces menos trabajo. El precio es la memoria: toda la pasada hacia delante debe guardarse hasta que el camino de vuelta llegue a cada resultado, algo que los puntos de control alivian recalculando algunos.

No construya nunca la tabla

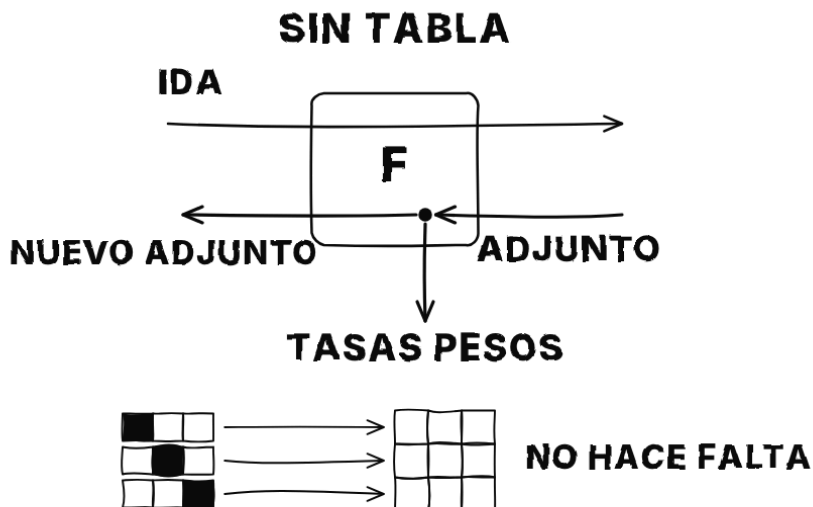
Dibuje en un papel una pequeña tabla de multiplicar. Arriba escriba dos números, dos y tres; en el lado izquierdo escriba otros dos, uno y cuatro. Llene cada casilla con el producto de su número de fila por su número de columna: la primera fila dice dos y tres, la segunda ocho y doce. Esa pequeña cuadrícula, que a un niño le lleva un minuto, es exactamente el conjunto de tasas que necesita un programa de entrenamiento para los pesos de una capa, y al final de esta parte verá por qué.



Una tabla de multiplicar, y entra fila, sale fila

Mire otra vez el camino de vuelta del modo inverso. En cada paso solo ocurren dos cosas: el adjunto, una sola fila, se multiplica por el jacobiano de los pesos del paso, y la misma fila se multiplica por el jacobiano de entrada del paso. A un paso nunca se le pide nada más. El libro da un nombre a estas dos operaciones: productos vector-jacobiano. Un producto vector-jacobiano es una función: usted le da una fila con un número por cada salida del paso, y ella le devuelve una fila con un número por cada entrada, o por cada peso. El modo directo tiene su propia operación en espejo, el producto jacobiano-vector, que para una función con una sola salida es la conocida tasa a lo largo de una dirección elegida.

Así que un paso hacia atrás no son más que dos llamadas. Llame al producto vector-jacobiano de los pesos con el adjunto, y salen las tasas de la puntuación respecto de los pesos de ese paso. Llame al producto vector-jacobiano de entrada con el mismo adjunto, y sale el nuevo adjunto, listo para llevarlo un paso atrás. La figura del libro muestra una primitiva como una caja: la pasada hacia delante entra por un lado y sale por el otro, y dos flechas hacia atrás, una por cada uno de estos productos, llevan el adjunto de vuelta a través de ella. Y aquí está el sentido de toda la sección: un sistema completo de diferenciación automática es un conjunto de primitivas, cada una emparejada con sus dos productos vector-jacobiano, y ningún jacobiano tiene que construirse nunca en memoria.

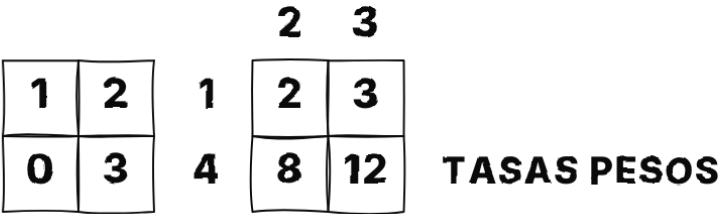


Una primitiva con dos reglas hacia atrás

Con esto no se pierde nada. Si de verdad quiere un jacobiano completo, puede llamar al producto una vez con una fila que tenga un uno en el primer lugar y ceros en los demás, luego con un uno en el segundo lugar, y así sucesivamente, y cada llamada da una fila de la tabla. Pero el programa de entrenamiento nunca quiere la tabla; solo quiere productos con adjuntos.

Ahora la recompensa, para las dos piezas de las que está hecha una capa totalmente conectada. Primero, una proyección lineal: la salida es la tabla de pesos multiplicada por la entrada. Su jacobiano de entrada es simplemente la propia tabla de pesos, pero su jacobiano de los pesos es un bloque de tres direcciones: una salida, una fila de los pesos y una columna de los pesos. El producto de entrada no necesita

ningún bloque: cada número de entrada reúne las casillas del adjunto de las salidas a las que alimentó, cada una ponderada por el peso que las conectaba, lo que equivale a pasar el adjunto hacia atrás por la tabla de pesos puesta de lado. Tome una tabla de pesos cuya primera fila es uno y dos y cuya segunda fila es cero y tres, y un adjunto de uno y cuatro. La primera entrada alimentó las salidas a través de los pesos uno y cero, así que reúne uno por uno más cuatro por cero, que es uno; la segunda las alimentó a través de dos y tres, así que reúne uno por dos más cuatro por tres, que es catorce.



TASAS ENTRADA: 1, 14

Los dos productos de un paso lineal

El producto de los pesos es aún más bonito, y es su tabla de multiplicar. La tasa de la puntuación respecto del peso de una fila y una columna dadas es la casilla del adjunto de esa fila por la casilla de

la entrada de esa columna. El razonamiento del libro en palabras: la puntuación vista desde este paso es una suma de productos, cada peso multiplicado por su casilla del adjunto y por su casilla de la entrada, así que empujar un peso mueve la puntuación exactamente su casilla del adjunto por su casilla de la entrada. Con entradas dos y tres y adjunto uno y cuatro, la cuadrícula es dos y tres, luego ocho y doce: la tabla que dibujó al principio. Así que cada vez que una proyección lineal se ejecuta hacia delante, la pasada hacia atrás devuelve el adjunto por sus pesos puestos de lado y rellena una tabla de multiplicar para sus pesos, y no aparece en ningún sitio un bloque de tres direcciones.

Segundo, una función de activación aplicada número a número, como la ReLU, que no tiene pesos, así que solo importa el producto de entrada. Cada salida depende solo de la entrada en el mismo lugar, así que el jacobiano de entrada es una tabla que es cero en todas partes salvo a lo largo de su diagonal, donde contiene la pendiente de la activación en cada entrada. Multiplicar una fila por una tabla así simplemente escala cada casilla del adjunto por la pendiente en el mismo lugar. Para la ReLU con entradas menos uno, dos y tres, las pendientes son cero, uno y uno; un adjunto de cinco, seis y siete vuelve como cero, seis y siete. Para una capa de mil números, la tabla completa tendría un millón de casillas, todas cero salvo mil, y el producto nunca la necesita. Así es como obtiene sus tasas la aplicación de fotos de su teléfono: cada operación de la biblioteca de entrenamiento lleva su propia pequeña regla hacia atrás de este tipo.

PENDIENTES ADJUNTO

-1	0			5	→	0
2		1		6	→	6
3			1	7	→	7

1000 × 1000: CASI TODO CEROS

La ReLU solo escala

Para recordar

El camino de vuelta solo multiplica una fila por un jacobiano, y estas operaciones son los productos vector-jacobiano, uno para la entrada y otro para los pesos de cada primitiva. Un sistema de diferenciación automática es un conjunto de primitivas, cada una con sus dos productos, y nunca construye un jacobiano, aunque podría hacerlo, fila a fila. En una proyección lineal, el producto de entrada devuelve el adjunto por la tabla de pesos puesta de lado, y el producto de los pesos es una tabla de multiplicar de casillas del adjunto y casillas de la entrada. En una activación aplicada número a número, el producto de entrada solo escala cada casilla del adjunto por la pendiente de la activación en la entrada correspondiente, igual que la ReLU convirtió cinco, seis y siete en cero, seis y siete.

Preguntas

1. ¿Por qué un sistema de diferenciación automática nunca necesita construir un jacobiano en memoria?
 - a. Porque el camino de vuelta solo multiplica una fila por un jacobiano, y cada primitiva puede hacerlo directamente con sus dos productos vector-jacobiano
 - b. Porque los jacobianos son siempre cero salvo en su diagonal
 - c. Porque las tasas se hallan empujando cada peso
 - d. Porque la pasada hacia delante ya guarda los jacobianos de cada paso

2. En una proyección lineal, ¿cuáles son las tasas de la puntuación respecto de sus pesos?
 - a. La propia tabla de pesos, puesta de lado
 - b. Las casillas del adjunto sumadas, una vez por cada peso
 - c. Una tabla de multiplicar: la casilla del adjunto de la fila del peso por la casilla de la entrada de su columna
 - d. Un bloque de tres direcciones que hay que construir antes de la pasada hacia atrás
3. ¿Qué le hace al adjunto el producto vector-jacobiano de entrada de la ReLU?
 - a. Suma la pendiente de la ReLU a cada casilla del adjunto
 - b. Multiplica cada casilla del adjunto por la pendiente de la ReLU en la entrada del mismo lugar
 - c. Sustituye el adjunto por las entradas de la capa
 - d. Multiplica el adjunto por la tabla completa de pendientes, un millón de casillas para mil números
4. Si de verdad quisiera el jacobiano de entrada completo de una primitiva, ¿cómo podría obtenerlo a partir de su producto vector-jacobiano?
 - a. Llamándolo una vez con una fila de unos
 - b. Llamándolo con los pesos en lugar del adjunto
 - c. Ejecutando dos veces la pasada hacia delante con un empujón
 - d. Llamándolo con una fila que tenga un uno en un lugar y ceros en los demás, una vez por cada lugar, una fila de la tabla por llamada

Respuestas

1. **a)** Un paso hacia atrás solo pide el adjunto por el jacobiano de los pesos y el adjunto por el jacobiano de entrada, así que bastan primitivas emparejadas con esos dos productos.
2. **c)** La puntuación vista desde el paso es una suma de productos de cada peso por su casilla del adjunto y su casilla de la entrada, así que empujar un peso mueve la puntuación esa casilla del adjunto por esa casilla de la entrada.
3. **b)** El jacobiano de entrada de una activación aplicada número a número es cero fuera de su diagonal, así que el producto simplemente escala cada casilla del adjunto por la pendiente del mismo lugar, sin construir ninguna tabla.
4. **d)** Cada llamada con una fila que tiene un solo uno da una fila de la tabla, así que repetirla para cada lugar reconstruye todo el jacobiano.

Ejemplos resueltos

Las tasas de los pesos de una proyección Una proyección lineal recibe las entradas uno y cinco, y el adjunto que vuelve es dos y tres. ¿Cuáles son las tasas de la puntuación respecto de sus cuatro pesos?

La tasa del peso de una fila y una columna dadas es la casilla del adjunto de la fila por la casilla de la entrada de la columna.

Primera fila, casilla del adjunto dos: dos por uno es dos, dos por cinco es diez.

Segunda fila, casilla del adjunto tres: tres por uno es tres, tres por cinco es quince.

Colóquelas como una cuadrícula, con las filas según las casillas del adjunto y las columnas según las entradas.

Respuesta: La primera fila es dos y diez; la segunda, tres y quince.

De vuelta a través de una ReLU Una capa ReLU recibió las entradas cuatro, menos dos, un medio y menos tres. El adjunto que vuelve es uno, dos, tres y cuatro. ¿Qué adjunto pasa?

La pendiente de la ReLU es uno donde la entrada es positiva y cero donde es negativa.

Así que las pendientes son uno, cero, uno y cero.

Escale cada casilla del adjunto por la pendiente del mismo lugar.

Respuesta: Uno, cero, tres y cero.

Ejercicios

1. Una proyección lineal tiene una tabla de pesos cuya primera fila es dos y uno y cuya segunda fila es uno y tres. El adjunto que vuelve es uno y dos. ¿Qué nuevo adjunto pasa el producto de entrada?

Pista: Cada entrada reúne las casillas del adjunto de las salidas a las que alimentó, cada una ponderada por el peso que las conectaba; la primera entrada alimentó las salidas a través de la primera casilla de cada fila.

Respuesta: Cuatro y siete: la primera entrada recibe uno por dos más dos por uno, que es cuatro; la segunda recibe uno por uno más dos por tres, que es siete.

2. Una primitiva tiene cinco salidas y ocho entradas. ¿Cuántas llamadas a su producto vector-jacobiano de entrada hacen falta para reconstruir todo su jacobiano de entrada, y cuántos números da cada llamada?

Pista: Cada llamada con un solo uno da una fila de la tabla; pregúntese cuántas filas tiene la tabla.

Respuesta: Cinco llamadas, una por cada salida, y cada una da una fila de ocho números, uno por entrada.

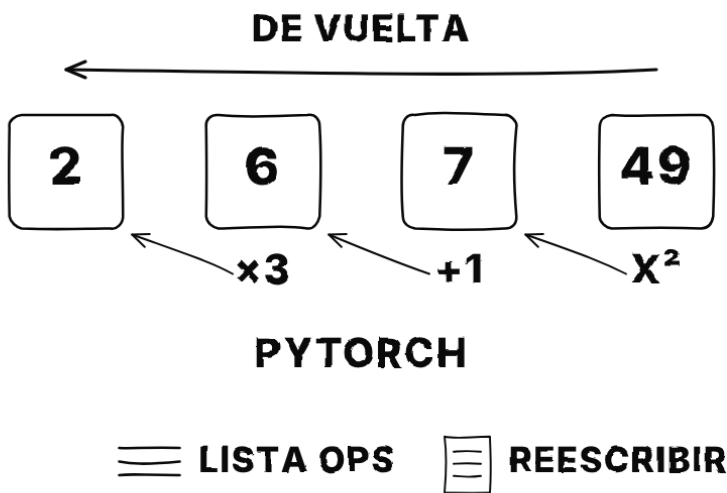
3. Una función de activación se aplica a una capa de quinientos números. ¿Cuántas casillas tendría su jacobiano de entrada completo, y cuántas de ellas serían cero?

Pista: La tabla tiene una fila por salida y una columna por entrada, y solo la diagonal no es cero.

Respuesta: Doscientas cincuenta mil casillas, de las que doscientas cuarenta y nueve mil quinientas son cero; solo las quinientas de la diagonal contienen pendientes, y por eso el producto se limita a escalar el adjunto.

Notas dejadas por el camino

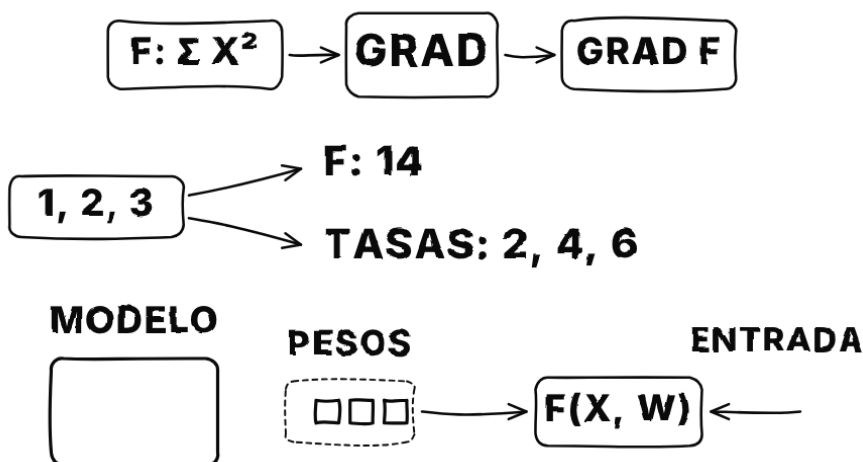
Haga la cadena de la calculadora una última vez, pero con notas adhesivas. Escriba el dos inicial en una nota. Cada vez que pulse un botón, escriba el resultado en una nota nueva y anote debajo qué lo produjo: «por tres, de la nota anterior», «más uno, de la nota anterior», «al cuadrado, de la nota anterior». Cuando llegue a cuarenta y nueve, puede encontrar el camino de vuelta sin recordar nada: cada nota le dice qué operación la produjo y de dónde venía. Ese rastro de notas es, casi literalmente, la forma en que PyTorch hace el modo inverso.



Un rastro de notas adhesivas

El libro dice primero que hay muchas maneras de construir un sistema en modo inverso, desde llevar una lista de cada operación realizada, como hace TensorFlow, hasta reescribir el propio código del programa en un segundo programa que calcula las tasas. Después examina los dos estilos que vimos antes, las funciones y los objetos.

En una biblioteca funcional como JAX todo es una función, y la diferenciación también. Un producto vector-jacobiano se convierte en una función que fabrica funciones. Usted le da una función y un punto que mirar; ella le devuelve una función nueva, que recibe una fila con un número por cada salida y devuelve esa fila multiplicada por el jacobiano en su punto, un número por cada entrada. Para una función con una sola salida hay algo aún más sencillo: dele la función, y le devuelve una segunda función que calcula el gradiente, las tasas respecto de cada entrada, en el punto que usted quiera. El ejemplo del libro es la suma de los cuadrados de diez números: la función gradiente, dados diez números, devuelve diez tasas. Pruébelo con tres números, uno, dos y tres: la suma de los cuadrados es catorce, y las tasas son dos, cuatro y seis, el doble de cada número, porque cada cuadrado crece el doble de rápido que su número. JAX ofrece estas funciones que fabrican funciones listas para usar, PyTorch copia la misma interfaz en un módulo propio, y el resultado puede compilarse en un grafo optimizado; combinar con astucia los dos tipos de productos da incluso segundas tasas, las tasas de las tasas.

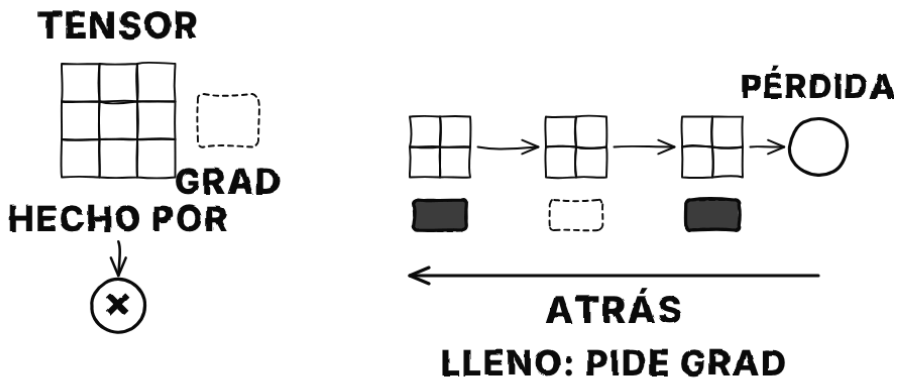


Una función que fabrica funciones

Sin embargo, los modelos reales se escriben casi siempre como objetos que guardan dentro sus pesos. Una salida es purificarlos: sacar los pesos del objeto a una colección sencilla y llamar al modelo como una función que recibe su entrada y esos pesos uno junto al otro, que es lo que pide el estilo funcional.

PyTorch también hace el trabajo directamente, con las notas adhesivas. Cada tensor, el nombre que PyTorch da a una tabla de números, lleva dos cosas más: un hueco vacío para su gradiente y una nota de la operación que lo produjo. La figura del libro lo muestra a la izquierda: un tensor, su hueco de gradiente vacío y una flecha hacia la operación de la que vino. Cuando usted pide la pasada hacia atrás sobre un solo número, como la pérdida de un lote de fotos, PyTorch

sigue las notas hacia atrás por el grafo computacional y aplica por el camino la regla hacia atrás de cada operación. A la derecha de la figura los huecos se llenan: los gradientes se guardan dentro de los propios tensores, pero solo en aquellos cuyo dueño ha dicho que necesitan un gradiente, para no gastar memoria en los demás.



MICROGRAD

Los tensores llevan sus propias notas

El libro reconoce que esto es solo un esbozo y remite a la documentación oficial para los detalles. Su consejo más cálido es que usted mismo construya un pequeño sistema en modo inverso, y nombra una de las muchas versiones didácticas que hay en internet, micrograd, que cabe en unos pocos archivos cortos.

Pruébalo

Escriba la cadena dos, por tres, más uno, al cuadrado en notas adhesivas, cada nota diciendo qué la produjo. Ahora empiece por la última nota y camine hacia atrás, escribiendo en cada nota cuánto se mueve el cuarenta y nueve final con ella: catorce en el siete, catorce en el seis, cuarenta y dos en el dos. Acaba de hacer a mano lo que hace PyTorch cuando usted le pide la pasada hacia atrás.

Para recordar

El modo inverso puede construirse de muchas maneras, desde una lista de operaciones hasta reescribir el código del programa. En el estilo funcional de JAX, diferenciar es una función que fabrica funciones: dada una función y un punto, devuelve el producto vector-jacobiano, y para una sola salida devuelve la función gradiente, igual que la suma de los cuadrados de uno, dos y tres tiene tasas dos, cuatro y seis. Los objetos que guardan dentro sus pesos pueden purificarse en funciones así. PyTorch, en cambio, hace que cada tensor lleve una nota de la operación que lo produjo y un hueco de gradiente vacío, recorre las notas hacia atrás cuando se le pide y llena los huecos de los tensores que pidieron gradientes.

Preguntas

1. En el estilo funcional de JAX, ¿qué recibe usted cuando pide el producto vector-jacobiano de una función en un punto?
 - a. El jacobiano completo de la función en ese punto, como una tabla
 - b. Una función nueva que recibe una fila con un número por cada salida y devuelve esa fila por el jacobiano
 - c. El valor de la función en ese punto, calculado dos veces
 - d. Una lista de las operaciones que realizó la función, para usarla más tarde
2. ¿Cómo sabe PyTorch recorrer un cálculo hacia atrás?
 - a. Vuelve a ejecutar la pasada hacia delante moviendo un poco cada peso
 - b. Reescribe el código del modelo en un segundo programa antes de empezar el entrenamiento
 - c. Cada tensor lleva una nota de la operación que lo produjo, y la pasada hacia atrás sigue esas notas
 - d. Pide al objeto modelo una fórmula de sus tasas
3. ¿Por qué PyTorch llena el hueco de gradiente solo en los tensores que han pedido un gradiente?
 - a. Para no gastar memoria en gradientes que nadie necesita
 - b. Porque los demás tensores no tienen ninguna operación que los produjera
 - c. Porque la pasada hacia atrás no puede atravesar tensores sin gradiente
 - d. Para que la pasada hacia delante vaya más rápido
4. ¿Qué significa purificar un modelo?

- a. Quitar las funciones de activación para que el modelo sea lineal
- b. Borrar los pesos cuyo gradiente es cero
- c. Compilar el modelo en un grafo optimizado
- d. Sacar los pesos del objeto para que el modelo pueda llamarse como una función de su entrada y de sus pesos

Respuestas

1. **b)** Allí diferenciar es una función que fabrica funciones: dada una función y un punto, devuelve una función nueva que multiplica cualquier fila por el jacobiano en ese punto.
2. **c)** Cada tensor guarda una nota de la operación que lo hizo, así que al pedir la pasada hacia atrás sobre un solo número PyTorch puede seguir las notas hacia atrás por el grafo.
3. **a)** Los gradientes se guardan dentro de los propios tensores, y solo donde su dueño dijo que hacían falta, para no gastar memoria en los demás.
4. **d)** Los objetos guardan dentro sus pesos; purificar los saca a una colección sencilla, para que el modelo pueda llamarse a la manera funcional, con la entrada y los pesos uno junto al otro.

Ejemplos resueltos

El gradiente de una suma de cuadrados El ejemplo del libro pide la función gradiente de la suma de los cuadrados de diez números. ¿Cuántos números devuelve esa función gradiente, y qué devuelve para los tres números tres, menos uno y dos, si usamos tres números en lugar de diez?

La función tiene una sola salida, así que su gradiente tiene una tasa por cada entrada.

Con diez números de entrada, la función gradiente devuelve diez tasas. Cada cuadrado crece el doble de rápido que su número, y los demás cuadrados no cambian cuando se mueve un número.

Así que cada tasa es el doble de su número: seis, menos dos y cuatro.

Respuesta: Diez números para diez entradas; para tres, menos uno y dos, las tasas son seis, menos dos y cuatro.

Ejercicios

1. Con la función gradiente de la suma de cuadrados, ¿cuáles son las tasas en los números un medio y cuatro?

Pista: ¿A qué velocidad crece un cuadrado comparado con su número?

Respuesta: Uno y ocho: cada tasa es el doble de su número.

2. Una función recibe seis números y da dos números. Usted pide a la biblioteca funcional su producto vector-jacobiano en algún punto. ¿Cuántos números recibe la función que le devuelven, y cuántos devuelve?

Pista: La fila que recibe tiene un número por cada salida de la función original.

Respuesta: Recibe dos números, uno por salida, y devuelve seis, uno por entrada: la fila multiplicada por el jacobiano de dos por seis en ese punto.

3. Explique con sus propias palabras cómo un rastro de notas adhesivas le permite hacer la pasada hacia atrás sin recordar el programa.

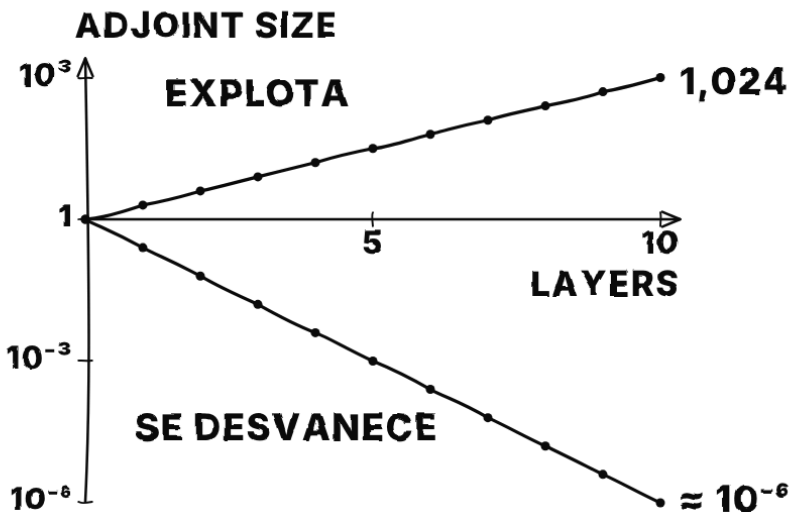
Pista: Piense en lo que dice cada nota sobre el origen de su número.

Respuesta: Cada nota registra su valor, la operación que lo hizo y la nota de la que vino. Partiendo del resultado final se pueden seguir esos enlaces hacia atrás y, en cada nota, aplicar la regla hacia atrás de esa operación

para saber cuánto se mueve con ella el resultado final. PyTorch hace lo mismo con las notas que adjunta a cada tensor.

Un cuarto, diez veces seguidas

Escriba uno en una calculadora y multiplique por un cuarto, diez veces seguidas. Quizá espere algo pequeño, pero no tan pequeño: la pantalla muestra más o menos una millonésima. Ahora bórrela, escriba otra vez uno y multiplique por dos, treinta veces seguidas: la pantalla pasa de mil millones. Estas dos series de pulsaciones son todo lo que contiene esta parte.



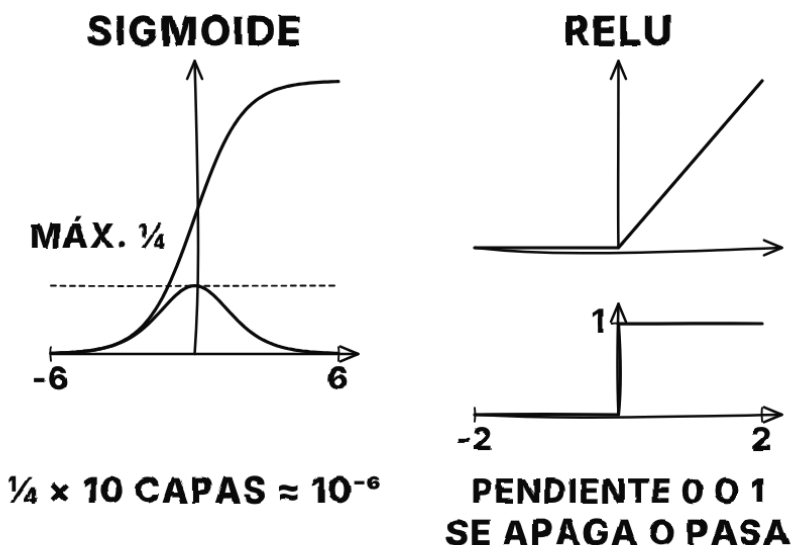
Encoger y crecer, capa a capa

Recuerde lo que hace una función de activación en la pasada hacia atrás: cada valor del adjunto se multiplica por la pendiente de la

activación en la entrada correspondiente. Una red profunda tiene una activación en cada capa, así que en su camino de vuelta el adjunto se multiplica por una pendiente así una y otra vez, una vez por capa. Si las pendientes son menores que uno en todas partes, el adjunto encoge con cada capa, como su cuarto, y tras suficientes capas las tasas de los primeros pesos son prácticamente cero: es el problema del desvanecimiento del gradiente. Si las pendientes son mayores que uno en todas partes, el adjunto crece con cada capa, como su duplicación, y las tasas se disparan hacia el infinito: el problema de la explosión del gradiente.

No son simples curiosidades, porque un ordenador guarda los números con una precisión limitada, normalmente treinta y dos cifras binarias o menos. Un número que encoge lo suficiente se redondea sin más a cero, y uno que crece lo suficiente ya no cabe, y con muchas capas esto ocurre enseguida. El libro añade un giro curioso: por estos diminutos efectos de redondeo, una pila de capas puramente lineales en un ordenador no es perfectamente lineal, y hasta se ha aprovechado para entrenar redes profundas sin ninguna activación.

La sigmoide muestra cómo aparece el desvanecimiento del gradiente. Era una activación habitual en el pasado, una versión suave de un escalón: una curva en forma de S que sube suavemente de cero a uno. Su pendiente en cualquier punto es igual a su valor por uno menos su valor. Como el valor está entre cero y uno, ese producto es máximo cuando el valor es un medio, y entonces es un medio por un medio, un cuarto; con un valor de nueve décimas es solo nueve centésimas. La figura del libro muestra la curva en S con su pendiente al lado, una joroba baja que nunca pasa de un cuarto. Así que cada capa sigmoide reduce el adjunto al menos a un cuarto, y diez de ellas lo reducen a más o menos una millonésima, su primera serie en la calculadora.



Pendientes de la sigmoide frente a pendientes de la ReLU

¿Podría una activación tener una pendiente exactamente igual a uno en todas partes? Solo una función que no hace nada, que devuelve su entrada sin cambios, y entonces las capas que la rodean se funden en una sola capa lineal. Necesitamos algo lo bastante lineal para que las pendientes no hagan daño, pero lo bastante curvado para mantener separadas las capas. La ReLU encaja: su pendiente es cero para las entradas negativas y uno para las positivas. Su figura muestra una línea tendida en cero que luego sube a cuarenta y cinco grados, y a su lado la pendiente, un escalón de cero a uno. Así que cada valor del adjunto o se apaga, lo que deja muchos ceros y ahorra trabajo, o pasa sin cambios, así que nada encoge ni explota.

Hay un premio para la memoria. La ReLU conserva los números positivos y convierte los negativos en cero, así que su salida indica igual de bien que su entrada dónde la pendiente es uno y dónde es cero. Por tanto, la pasada hacia atrás puede trabajar a partir de la salida, y no hace falta conservar la entrada: la capa puede sobrescribirla en la pasada hacia delante, ahorrando algo de memoria sin estropear ninguna tasa, y PyTorch ofrece exactamente esto como opción in situ.

Para recordar

Cada activación multiplica el adjunto por su pendiente en el camino de vuelta, así que en una red profunda esas pendientes se multiplican entre sí. Las pendientes menores que uno hacen que las tasas de las primeras capas se desvanezcan, las mayores que uno hacen que exploten, y la precisión limitada de los ordenadores convierte ambas cosas en fallos reales. La pendiente de la sigmoide nunca pasa de un cuarto, así que diez capas sigmoide pueden reducir el adjunto a una millonésima. La pendiente de la ReLU es cero o uno, así que el adjunto o se apaga o pasa sin cambios, y como su salida muestra dónde la pendiente es uno, su entrada puede sobrescribirse para ahorrar memoria.

Preguntas

1. ¿Por qué una red profunda con activaciones sigmoide corre el riesgo de que el gradiente se desvanezca?

- a. Porque la salida de la sigmoide puede ser mayor que uno
 - b. Porque la pendiente de la sigmoide nunca pasa de un cuarto, y esas pendientes se multiplican capa tras capa
 - c. Porque la sigmoide apaga todas las entradas negativas
 - d. Porque la sigmoide no tiene pendiente en cero
2. ¿Por qué el desvanecimiento y la explosión del gradiente son fallos reales y no simplemente números muy pequeños o muy grandes?
- a. Porque entonces las tasas apuntan en la dirección equivocada
 - b. Porque la pérdida deja de ser un solo número
 - c. Porque los ordenadores guardan los números con una precisión limitada, así que los números diminutos se vuelven cero y los enormes ya no caben
 - d. Porque entonces hay que ejecutar dos veces la pasada hacia delante
3. ¿Por qué no usar una activación cuya pendiente sea exactamente uno en todas partes?
- a. Porque solo la función que devuelve su entrada sin cambios tiene esa pendiente, y entonces las capas que la rodean se funden en una sola capa lineal
 - b. Porque una función así haría explotar todos los gradientes
 - c. Porque su pendiente sería cero para las entradas negativas
 - d. Porque los ordenadores no pueden guardar exactamente el número uno

4. ¿Por qué una capa ReLU puede sobrescribir su entrada durante la pasada hacia delante?
- a. Porque la ReLU no tiene ninguna regla hacia atrás
 - b. Porque ninguna capa vuelve a necesitar nunca la entrada
 - c. Porque sobrescribir hace más exactas las tasas
 - d. Porque su salida indica igual de bien que su entrada dónde la pendiente es uno y dónde es cero

Respuestas

1. **b)** En el camino de vuelta, el adjunto se multiplica por la pendiente de la activación en cada capa, y la pendiente de la sigmoide es como mucho un cuarto, así que diez capas pueden reducirlo a más o menos una millonésima.
2. **c)** Los números se guardan con treinta y dos cifras binarias o menos, así que un número que encoge lo suficiente se redondea a cero y uno que crece lo suficiente ya no cabe.
3. **a)** La única función con pendiente uno en todas partes no hace nada, y sin una curva las capas lineales se funden en una; la ReLU es lo bastante lineal pero sigue teniendo un doblez.
4. **d)** La ReLU conserva los números positivos y pone a cero los negativos, así que la pasada hacia atrás puede leer las pendientes en la salida, y la entrada puede sobrescribirse para ahorrar memoria.

Ejemplos resueltos

La pendiente máxima de la sigmoide La pendiente de la sigmoide es igual a su valor por uno menos su valor. ¿Cuál es la pendiente cuando el valor es un medio, y cuando es ocho décimas? ¿Y por cuánto pueden escalar, como mucho, cinco capas sigmoide un valor del adjunto?

Con el valor un medio: un medio por un medio es un cuarto.

Con el valor ocho décimas: ocho décimas por dos décimas son dieciséis centésimas, menos de un cuarto.

La pendiente es máxima en un medio, así que nunca pasa de un cuarto.

A través de cinco capas, el valor del adjunto se multiplica como mucho cinco veces por un cuarto: un cuarto de un cuarto de un cuarto de un cuarto de un cuarto es una parte entre mil veinticuatro.

Respuesta: Un cuarto y dieciséis centésimas; cinco capas reducen el valor del adjunto, como mucho, a más o menos una milésima.

Ejercicios

1. ¿Cuál es la pendiente de la sigmoide cuando su valor es dos décimas, y cuando es noventa y nueve centésimas?

Pista: Multiplique el valor por uno menos el valor.

Respuesta: Dieciséis centésimas, y más o menos una centésima: dos décimas por ocho décimas, y noventa y nueve centésimas por una centésima, que son noventa y nueve diezmilésimas.

2. Un valor del adjunto igual a uno vuelve a través de tres capas cuyas pendientes de activación son cada una uno y medio. ¿En qué se convierte? ¿Y en qué se convertiría a través de tres capas ReLU cuyas entradas fueran todas positivas?

Pista: En el camino de vuelta, cada capa multiplica el valor del adjunto por su pendiente.

Respuesta: Tres y tres octavos, porque uno y medio tres veces seguidas es tres coma tres siete cinco, el comienzo de una explosión; a través de tres capas ReLU con entradas positivas sigue siendo uno, porque cada pendiente es uno.

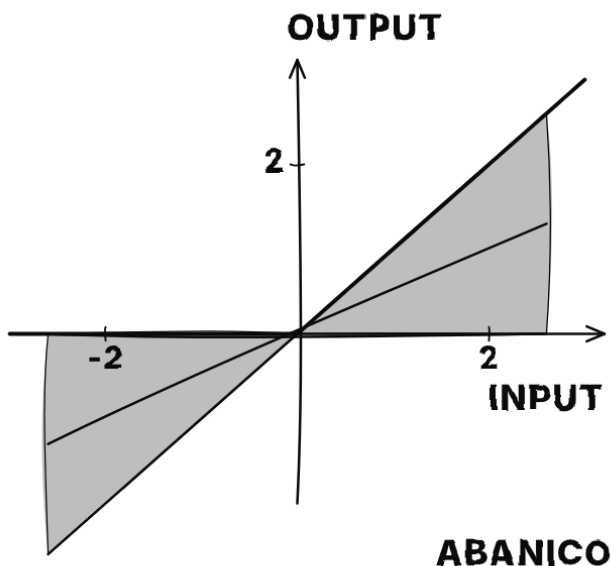
3. Explique con sus propias palabras por qué la ReLU evita tanto el desvanecimiento como la explosión del gradiente en los valores que deja pasar.

Pista: ¿Cuáles son las dos únicas pendientes que puede tener la ReLU?

Respuesta: La pendiente de la ReLU es cero para las entradas negativas y uno para las positivas. Cada valor o se apaga o se multiplica exactamente por uno, así que los valores que pasan nunca encogen ni crecen, por muchas capas que haya.

Una regla en la esquina

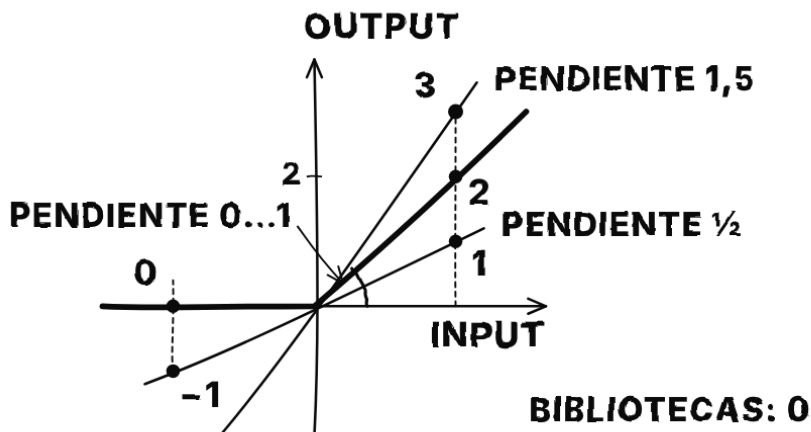
Dibuje la ReLU en una hoja de papel: una línea tendida sobre el eje horizontal a la izquierda y, desde la esquina en cero, una línea que sube a cuarenta y cinco grados hacia la derecha. Ahora ponga un lápiz que pase por la esquina e inclínelo despacio. En horizontal, queda por debajo del dibujo en todas partes; inclinado hasta cuarenta y cinco grados, sigue quedando por debajo; inclinado más, corta el brazo que sube, e inclinado hacia abajo, corta el brazo plano de la izquierda. Así que en la esquina no hay una pendiente, sino todo un abanico, todas las inclinaciones desde la horizontal hasta cuarenta y cinco grados.



Un lápiz inclinado en la esquina

Este es el pequeño detalle que el libro ha dejado para ahora. La ReLU no tiene una pendiente ordinaria en cero, así que una red hecha de ReLU no es suave en todas partes, y las reglas de las partes anteriores parecen romperse en ese punto. La respuesta práctica es que casi nunca importa. El entrenamiento empieza con pesos aleatorios y avanza en pequeños pasos aleatorios, así que la probabilidad de que una entrada caiga exactamente en cero es prácticamente nula, y un pelo a cualquiera de los dos lados la pendiente está perfectamente definida.

La respuesta más cuidadosa usa el abanico que acaba de dibujar. Para una función que se curva hacia arriba, una función convexa, un subgradiente en un punto es la pendiente de una recta que pasa por ese punto y queda por debajo de toda la función en todas partes. Donde la función es suave solo existe una recta así, la tangente de siempre, y su pendiente es la tasa de siempre. En una esquina existen muchas rectas así, y el conjunto de sus pendientes se llama subdiferencial. Para la ReLU, el subdiferencial es cero para las entradas negativas, uno para las positivas, y en cero es cada número de cero a uno. Compruebe una de ellas: tome la recta por la esquina con pendiente un medio. En la entrada dos da uno, mientras que la ReLU da dos, así que la recta queda por debajo; en menos dos da menos uno, mientras que la ReLU da cero, así que vuelve a quedar por debajo. Una pendiente de uno y medio no sirve: en la entrada dos da tres, por encima del dos de la ReLU. En la práctica, la mayoría de las bibliotecas simplemente eligen cero como pendiente de la ReLU en cero, y seguir en cada paso del entrenamiento un subgradiente elegido se llama descenso por subgradiente.



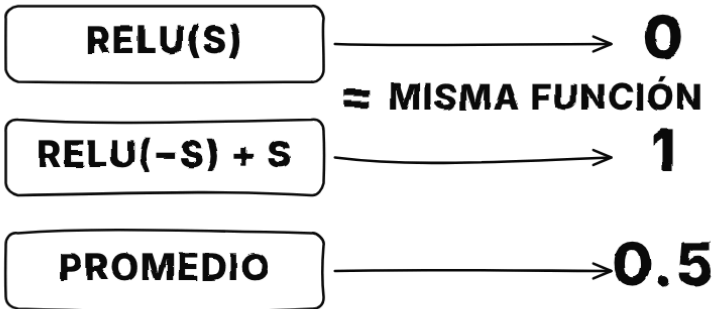
Comprobar un subgradiente

El libro reconoce que la historia es aún más delicada. Para funciones que no se curvan hacia arriba, puede no existir ningún subgradiente, y los matemáticos usan versiones más flexibles que solo miran un pequeño entorno del punto, como el subdiferencial de Clarke. Y la propia diferenciación automática puede tropezar aquí, porque dos maneras de escribir la misma función pueden dar pendientes distintas en la esquina. El ejemplo del libro: tome la ReLU de menos el número y vuelva a sumarle el número; para cualquier número, esto da otra vez exactamente la ReLU. Promedie eso con la ReLU sin más y sigue teniendo la ReLU. Sin embargo, en cero la pasada hacia atrás de PyTorch da cero para la primera, uno para la segunda y un medio para la tercera: tres versiones de la misma función, tres respuestas, y

por eso demostrar que estos programas son correctos exige reglas de la cadena más refinadas.

○ **CLARKE**

PENDIENTE EN 0



Una función, tres respuestas

El capítulo termina enviándole a practicar. Reconstruya un pequeño sistema en modo inverso, por ejemplo estudiando micrograd, que trabaja con números sueltos; en cuanto el grafo se ramifica y se junta, hay que poner antes sus pasos en un orden que respete quién alimenta a quién, un paso de ordenación barato, o la pasada hacia atrás seguirá caminos equivocados. Después escriba una primitiva propia en PyTorch, dando su cálculo hacia delante y sus productos vector-jacobiano, por ejemplo una de las activaciones entrenables del capítulo anterior, aunque PyTorch podría deducir solo su pasada hacia atrás. En JAX existen los mismos pasos: una versión didáctica

de JAX llamada autodidax, reglas hacia atrás propias para primitivas nuevas y un recetario de usos avanzados como las segundas tasas.

Aquí se cierra el capítulo. Desde una cadena de calculadora y su cuarenta y dos, usted ha seguido las tasas de un programa de entrenamiento hacia delante y hacia atrás, ha visto por qué gana el recorrido hacia atrás, por qué nunca construye sus tablas, y cómo se tratan las pendientes inofensivas de la ReLU e incluso su única esquina afilada.

Para recordar

La ReLU tiene una esquina en cero, pero en el entrenamiento una entrada casi nunca cae exactamente ahí, así que la pendiente ordinaria casi siempre está definida. Un subgradiente de una función convexa es la pendiente de una recta que pasa por un punto y queda por debajo de toda la función; en una esquina, su conjunto, el subdiferencial, es para la ReLU en cero cada pendiente de cero a uno, y las bibliotecas suelen elegir cero. Para otras funciones los subgradientes pueden no existir, y maneras equivalentes de escribir la ReLU hacen que PyTorch devuelva cero, uno o un medio en cero, así que la esquina exige cuidado en las demostraciones. El libro termina animando a practicar: reconstruir un pequeño sistema como micrograd, ordenando los pasos de un grafo ramificado, y escribir primitivas con sus propios productos vector-jacobiano en PyTorch o en JAX.

Preguntas

1. ¿Por qué la esquina de la ReLU en cero casi nunca importa en la práctica?
 - a. Porque las bibliotecas eliminan todos los ceros de las entradas antes de entrenar
 - b. Porque el entrenamiento empieza con pesos aleatorios y avanza en pequeños pasos aleatorios, así que una entrada casi nunca cae exactamente en cero
 - c. Porque la pendiente de la ReLU en cero es siempre uno
 - d. Porque la pasada hacia atrás se salta por completo las capas ReLU
2. ¿Qué pendientes son subgradientes de la ReLU en cero?
 - a. Solo cero, el valor que eligen las bibliotecas
 - b. Solo un medio, a medio camino entre los dos brazos
 - c. Cada pendiente de cero a uno, porque esas rectas por la esquina quedan por debajo de toda la ReLU
 - d. Cada pendiente de menos uno a uno
3. ¿Qué muestra el ejemplo del libro de las tres maneras de escribir la ReLU?
 - a. Que maneras equivalentes de escribir la misma función pueden hacer que la diferenciación automática devuelva pendientes distintas en la esquina
 - b. Que dos de las tres maneras no son realmente la ReLU
 - c. Que PyTorch calcula mal la ReLU para las entradas positivas
 - d. Que para la ReLU no existen subgradientes
4. Cuando el grafo de un programa se ramifica y se junta, ¿qué debe hacer un pequeño sistema en modo inverso como

micrograd antes de la pasada hacia atrás?

- a. Sustituir cada ReLU por una sigmoide
- b. Ejecutar dos veces la pasada hacia delante para comprobar los resultados
- c. Construir el jacobiano completo de cada paso
- d. Poner los pasos en un orden que respete quién alimenta a quién, un paso de ordenación barato

Respuestas

1. **b)** Con pesos iniciales aleatorios y pequeños pasos aleatorios, la probabilidad de que una entrada caiga exactamente en cero es prácticamente nula, y un pelo a cualquiera de los dos lados la pendiente está bien definida.
2. **c)** Una recta por la esquina queda por debajo de la ReLU en todas partes exactamente cuando su pendiente está entre la horizontal y los cuarenta y cinco grados, así que el subdiferencial en cero es cada número de cero a uno.
3. **a)** Las tres son la misma función, pero la pasada hacia atrás de PyTorch da cero, uno y un medio en cero, así que la esquina exige reglas de la cadena refinadas en las demostraciones.
4. **d)** En cuanto el grafo se ramifica y se junta, sus pasos deben ordenarse para que la pasada hacia atrás no siga caminos equivocados; micrograd lo hace con una ordenación barata.

Ejemplos resueltos

Tres ReLU, tres pendientes La segunda versión es la ReLU de menos el número, más el número; la tercera es el promedio de la ReLU sin más y la segunda versión. Compruebe que la segunda versión es igual a la ReLU en tres y en menos tres, y halle las

pendientes que da en cero la pasada hacia atrás de PyTorch, sabiendo que toma como cero la pendiente de la ReLU sin más en cero.

En tres: la ReLU de menos tres es cero, más tres da tres, lo mismo que la ReLU de tres.

En menos tres: la ReLU de tres es tres, más menos tres da cero, lo mismo que la ReLU de menos tres.

En cero, la pendiente de la ReLU sin más se toma como cero.

La pendiente de la segunda versión es menos la pendiente de la ReLU, por el signo menos interior, más uno por el número sumado: menos cero más uno es uno.

La tercera versión es el promedio de las dos, así que su pendiente es el promedio de cero y uno.

Respuesta: La segunda versión es igual a la ReLU en los dos puntos; las pendientes en cero son cero, uno y un medio.

Comprobar un subgradiente ¿Es una pendiente de un cuarto un subgradiente de la ReLU en cero? Pruebe la recta por la esquina con esa pendiente en las entradas cuatro y menos cuatro.

En cuatro, la recta da un cuarto de cuatro, que es uno; la ReLU da cuatro, así que la recta queda por debajo.

En menos cuatro, la recta da menos uno; la ReLU da cero, así que la recta vuelve a quedar por debajo.

Una recta por la esquina con pendiente entre cero y uno queda por debajo en los dos brazos, así que queda por debajo en todas partes.

Respuesta: Sí, un cuarto es un subgradiente de la ReLU en cero.

Ejercicios

1. ¿Es una pendiente de ocho décimas un subgradiente de la ReLU en cero? Compruebe la recta por la esquina en las entradas cinco y menos cinco.

Pista: Compare el valor de la recta con el de la ReLU en cada entrada.

Respuesta: Sí: en cinco la recta da cuatro, por debajo del cinco de la ReLU; en menos cinco da menos cuatro, por debajo del cero de la ReLU; y ocho décimas está entre cero y uno.

2. ¿Es una pendiente de menos un medio un subgradiente de la ReLU en cero? Encuentre una entrada donde la recta por la esquina suba por encima de la ReLU.

Pista: Mire a la izquierda, donde la ReLU está tendida sobre el cero.

Respuesta: No: en menos dos la recta da uno, por encima del cero de la ReLU, así que no queda por debajo de toda la función.

3. Explique con sus propias palabras qué es un subgradiente y por qué un punto suave solo tiene uno.

Pista: Piense en un lápiz que pasa por el punto y se inclina.

Respuesta: Para una función que se curva hacia arriba, un subgradiente en un punto es la pendiente de una recta que pasa por ese punto y queda por debajo de toda la función. En un punto suave solo lo consigue la tangente, así que el único subgradiente es la pendiente ordinaria; en una esquina el lápiz puede inclinarse en todo un abanico de posiciones, lo que da muchos subgradientes.