

由以下网站提供：explained.ai

深度学习所需的矩阵微积分

完整中文译本 · 原作者：Terence Parr、Jeremy Howard

正文、表格文字、推导说明及附录均已翻译；公式和图内英文保留原样。本文中的“我们”指原作者。原文的符号约定与推导保持不变，未将译文改写为新的数学教材。

原文：[The Matrix Calculus You Need For Deep Learning](#) · 翻译与排版：2026-10-10

Terence Parr 与 Jeremy Howard

(以下简介沿用原文：Terence 是 Google 的技术负责人，曾任[旧金山大学数据科学硕士项目](#)的计算机 / 数据科学教授。你也许知道他是 ANTLR 解析器生成器的作者。更多资料可参见 Jeremy 的[fast.ai](#) 课程，以及旧金山大学数据研究所的[线下深度学习课程](#)。)

如有评论、建议或纠错，请联系 [Terence](#)。

[可打印版本](#) (此 HTML 由 [bookish](#) 根据标记文本生成)。原站也提供一份[中文版本](#) (其内容未经原作者核实)。

摘要

本文尝试讲清理解深度神经网络训练所需的全部矩阵微积分知识。我们只假设你掌握了微积分入门课程的内容，并在需要时提供链接，帮助你复习必要的数学知识。请注意：在开始实践、学习训练和使用深度学习模型之前，你不必先读懂本文。本文面向已经熟悉神经网络基础、希望进一步理解其数学原理的读者。如果读到某处卡住了，不必担心：回头重读上一节，试着写下几个例子，亲手推算。如果仍有疑问，我们很乐意在 [forums.fast.ai](#) 的 [Theory](#) 板块回答。注意：文末的[速查部分](#)汇总了本文讨论的关键矩阵微积分法则和术语。

目录

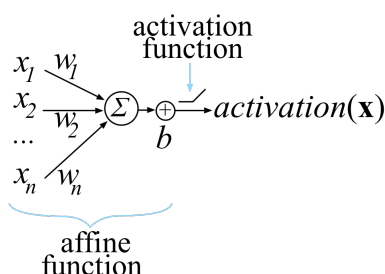
引言	3
回顾：标量求导法则	4
向量微积分与偏导数入门	5
矩阵微积分	5
雅可比矩阵的推广	6
向量逐元素二元运算的导数	8
涉及标量扩展的导数	10
向量求和归约	11
链式法则	12
单变量链式法则	12
单变量全导数链式法则	15
向量链式法则	17
神经元激活值的梯度	19
神经网络损失函数的梯度	22
对权重的梯度	22
对偏置的导数	23
总结	25
矩阵微积分速查	26
梯度与雅可比矩阵	26
向量的逐元素运算	26
标量扩展	27
向量归约	27
链式法则	27
符号说明	28
参考资源	28

引言

我们大多数人上一次接触微积分还是在学校里，但导数是机器学习的关键组成部分。深度学习尤其如此：它们通过优化损失函数来训练。随手翻开一篇机器学习论文，或 [PyTorch](#) 这样的库的文档，微积分就会像逢年过节的远房亲戚一样，突然闯回你的生活。而且，出现的还不只是普通的标量微积分：你需要的是把线性代数与多元微积分拉到一起的矩阵微分学。

不过……“需要”也许不太准确。Jeremy 的课程表明，借助现代深度学习库内置的自动微分，只掌握少量标量微积分知识，也能成为一流的深度学习实践者。但是，如果你真的想理解这些库底层发生了什么，并读懂讨论最新模型训练技术的学术论文，就需要了解矩阵微积分中的一部分知识。

例如，神经网络中单个计算单元的激活值，通常先由连接权重向量 $\mathbf{w}$ 与输入向量 $\mathbf{x}$ 的点积（线性代数中的运算），再加上一个标量偏置（阈值）得到： $z(\mathbf{x}) = \sum_i^n w_i x_i + b = \mathbf{w} \cdot \mathbf{x} + b$ 。函数 $z(\mathbf{x})$ 称为该单元的仿射函数，其后接一个整流线性单元，将负值截断为零： $\max(0, z(\mathbf{x}))$ 。这样的计算单元有时称为“人工神经元”，如下图所示：



神经网络由许多这样的单元组成，神经元被组织成多个称为层的集合。一层中各单元的激活值，成为下一层各单元的输入。最后一层中一个或多个单元的激活值，就是网络输出。

训练这个神经元，意味着选择权重 $\mathbf{w}$ 和偏置 b ，使其对全部 N 个输入 $\mathbf{x}$ 都产生期望输出。为此，我们最小化一个损失函数：它对所有输入向量 $\mathbf{x}$ ，比较网络最终的 $activation(\mathbf{x})$ 与 $target(\mathbf{x})$ ($\mathbf{x}$ 的期望输出)。为了最小化损失，我们使用某种梯度下降方法，例如普通的[随机梯度下降](#)（SGD）、带动量的SGD，或[Adam](#)。这些方法都需要 $activation(\mathbf{x})$ 对模型参数 $\mathbf{w}$ 和 b 的偏导数（梯度）。我们的目标是逐步调整 $\mathbf{w}$ 和 b ，使所有输入 $\mathbf{x}$ 上的总体损失不断减小。

只要仔细一些，就可以对常见损失函数（均方误差）的标量形式求导，从而得到梯度：

$$\frac{1}{N} \sum_{\mathbf{x}} (target(\mathbf{x}) - activation(\mathbf{x}))^2 = \frac{1}{N} \sum_{\mathbf{x}} (target(\mathbf{x}) - \max(0, \sum_i^{|\mathbf{x}|} w_i x_i + b))^2$$

但这还只是一个神经元，而神经网络必须同时训练所有层中全部神经元的权重和偏置。由于存在多个输入，以及可能存在的多个网络输出，我们确实需要函数对向量求导的一般法则，甚至需要向量值函数对向量求导的法则。

本文将逐步推导一些重要的向量偏导数计算法则，尤其是那些对训练神经网络有用的法则。这个领域称为矩阵微积分。好消息是，我们只需要其中很小的一部分，也就是本文介绍的内容。网上关于多元微积分和线性代数的资料很多，但它们通常作为两门独立的本科课程讲授，因此大多数资料也将二者分开讨论。真正讨论矩阵微积分的网页，往往只罗列法则而缺少解释，或者只讲了部分内容。由于符号密集，对基础概念的讨论又很少，除了少数数学专业读者之外，其他人通常很难读懂。（文末有一份附带评述的资源清单。）

这里，我们会通过重新推导、重新发现一些关键法则来解释矩阵微积分。你会发现，它其实并没有那么难！需要学习的不是几十条新规则，而只是几个关键概念。希望这篇短文能帮助你快速入门与神经网络训练相关的矩阵微积分。我们假设你已经熟悉神经网络结构和训练的基础知识；如果还不熟悉，请先完成 [Jeremy 课程](#) 的第一部分，再回来阅读。（请注意，与许多更偏学术的学习路径不同，我们强烈建议先在实践中学习训练和使用神经网络，再研究背后的数学。有了具体背景，数学会容易理解得多；而且，成为有效的实践者，并不需要先彻底掌握这些微积分知识。）

关于符号的说明：Jeremy 的课程完全使用代码而非数学符号解释概念，因为代码中不熟悉的函数很容易检索和实验。本文则相反，会使用大量数学符号，因为目标之一就是帮助你理解深度学习论文和书籍中的记号。[文末](#)有一张简明符号表，同时列出了可用于进一步搜索的词语或短语。

回顾：标量求导法则

希望你还记得下面这些主要的标量求导法则。如果记忆有些模糊，可以看看[可汗学院关于标量求导法则的视频](#)。

法则	$f(x)$	对 x 的标量导数记法	示例
常数	c	0	$\frac{d}{dx}99 = 0$
常数倍	cf	$c\frac{df}{dx}$	$\frac{d}{dx}3x = 3$
幂法则	x^n	nx^{n-1}	$\frac{d}{dx}x^3 = 3x^2$
和法则	$f + g$	$\frac{df}{dx} + \frac{dg}{dx}$	$\frac{d}{dx}(x^2 + 3x) = 2x + 3$
差法则	$f - g$	$\frac{df}{dx} - \frac{dg}{dx}$	$\frac{d}{dx}(x^2 - 3x) = 2x - 3$
乘积法则	fg	$f\frac{dg}{dx} + \frac{df}{dx}g$	$\frac{d}{dx}x^2x = x^2 + x2x = 3x^2$
链式法则	$f(g(x))$	$\frac{df(u)}{du}\frac{du}{dx}$ ，令 $u = g(x)$	$\frac{d}{dx}\ln(x^2) = \frac{1}{x^2}2x = \frac{2}{x}$

此外还有三角函数、指数函数等的求导法则，可以在[可汗学院微分学课程](#)中找到。

对于只有一个参数的函数 $f(x)$ ，你经常会看到用 f' 和 $f'(x)$ 简写 $\frac{d}{dx}f(x)$ 。我们不推荐这种记法，因为它没有明确指出求导所针对的变量。

可以把 $\frac{d}{dx}$ 看成一个算子：它把一个单参数函数映射到另一个函数。也就是说， $\frac{d}{dx}f(x)$ 将 $f(x)$ 映射为它对 x 的导数，也就是 $\frac{df(x)}{dx}$ 。同样，如果 $y = f(x)$ ，那么 $\frac{dy}{dx} = \frac{df(x)}{dx} = \frac{d}{dx}f(x)$ 。把求导视为一种算子，有助于化简复杂的导数，因为这个算子具有分配性质，并允许我们把常数提出去。例如，在下面的等式中，我们可以提出常数 9，再把求导算子分别作用到括号中的各项上。

$$\frac{d}{dx}9(x + x^2) = 9\frac{d}{dx}(x + x^2) = 9(\frac{d}{dx}x + \frac{d}{dx}x^2) = 9(1 + 2x) = 9 + 18x$$

这样一来，对 $9(x + x^2)$ 求导就转化成了一点算术运算，以及对 x 和 x^2 求导，后两者比原式容易处理得多。

向量微积分与偏导数入门

神经网络层并不是单个单参数函数 $f(x)$ 。因此，接下来看看具有多个参数的函数，例如 $f(x, y)$ 。比如， xy （即 x 与 y 相乘）的导数是什么？换句话说，稍微改变变量时，乘积 xy 会怎样变化？答案取决于改变的是 x 还是 y 。我们每次只对一个变量（参数）求导，因此这个双参数函数有两个不同的偏导数（分别对 x 和 y ）。偏导数使用的算子是 $\frac{\partial}{\partial x}$ ，而不是 $\frac{d}{dx}$ ；其中的符号是变体的 d ，不是希腊字母 δ 。所以， $\frac{\partial}{\partial x}xy$ 和 $\frac{\partial}{\partial y}xy$ 就是 xy 的偏导数，英文常简称为 partials。对于单参数函数，算子 $\frac{\partial}{\partial x}$ 与 $\frac{d}{dx}$ 等价（假定函数足够光滑）。不过，最好使用 $\frac{d}{dx}$ ，明确表示这里指的是标量导数。

对 x 求偏导，其实就是普通的标量求导，只需将式中的其他变量都视为常数。考虑函数 $f(x, y) = 3x^2y$ 。它对 x 的偏导数记作 $\frac{\partial}{\partial x}3x^2y$ 。从 $\frac{\partial}{\partial x}$ 的角度看，有三个常数：3、2 和 y 。因此， $\frac{\partial}{\partial x}3x^2y = 3y\frac{\partial}{\partial x}x^2 = 3y2x = 6yx$ 。对 y 求偏导时，则把 x 当作常数： $\frac{\partial}{\partial y}3x^2y = 3x^2\frac{\partial}{\partial y}y = 3x^2\frac{\partial}{\partial y}y = 3x^2 \times 1 = 3x^2$ 。继续阅读之前，最好自己推导一下，否则后文可能难以理解。如需帮助，可以观看[可汗学院的偏导数视频](#)。

为了说明我们讨论的是向量微积分，而不只是多元微积分，来看看如何处理刚才为 $f(x, y) = 3x^2y$ 算出的偏导数 $\frac{\partial f(x, y)}{\partial x}$ 和 $\frac{\partial f(x, y)}{\partial y}$ （也就是 $\frac{\partial}{\partial x}f(x, y)$ 和 $\frac{\partial}{\partial y}f(x, y)$ 的另一种写法）。不要让它们零散地放着，而是将它们组织成一个行向量。这个向量称为 $f(x, y)$ 的梯度，记为：

$$\nabla f(x, y) = \left[\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right] = [6yx, 3x^2]$$

所以， $f(x, y)$ 的梯度就是由其偏导数组成的向量。梯度属于向量微积分的范畴，处理的是把 n 个标量参数映射为一个标量的函数。现在再大胆一点，考虑同时对多个函数求导。

矩阵微积分

从一个函数的导数转向多个函数的导数，我们就从向量微积分进入了矩阵微积分。来计算两个函数的偏导数，每个函数都有两个参数。保留上一节的 $f(x, y) = 3x^2y$ ，再引入 $g(x, y) = 2x + y^8$ 。 g 的梯度有两个分量，每个参数对应一个偏导数：

$$\frac{\partial g(x, y)}{\partial x} = \frac{\partial 2x}{\partial x} + \frac{\partial y^8}{\partial x} = 2\frac{\partial x}{\partial x} + 0 = 2 \times 1 = 2$$

以及

$$\frac{\partial g(x, y)}{\partial y} = \frac{\partial 2x}{\partial y} + \frac{\partial y^8}{\partial y} = 0 + 8y^7 = 8y^7$$

由此得到梯度 $\nabla g(x, y) = [2, 8y^7]$ 。

梯度向量将一个特定标量函数的所有偏导数组织在一起。如果有两个函数，也可以把它们梯度逐行堆叠成矩阵。这样得到的就是雅可比矩阵（Jacobian matrix，简称 Jacobian），其中每一行都是一个梯度：

$$J = \begin{bmatrix} \nabla f(x, y) \\ \nabla g(x, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} & \frac{\partial f(x, y)}{\partial y} \\ \frac{\partial g(x, y)}{\partial x} & \frac{\partial g(x, y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 6yx & 3x^2 \\ 2 & 8y^7 \end{bmatrix}$$

欢迎来到矩阵微积分的世界！

注意，雅可比矩阵有多种表示方式。我们使用所谓的**分子布局**，但许多论文和软件采用分母布局。后者就是分子布局雅可比矩阵的转置（沿主对角线翻转）：

$$\begin{bmatrix} 6yx & 2 \\ 3x^2 & 8y^7 \end{bmatrix}$$

雅可比矩阵的推广

到目前为止，我们看的是雅可比矩阵的具体例子。为了给出更一般的定义，先把多个参数合并成一个向量参数： $f(x, y, z) \Rightarrow f(\mathbf{x})$ 。（文献中有时也用 $\vec{x}$ 表示向量。）像 $\mathbf{x}$ 这样的粗体小写字母表示向量，像 x 这样的斜体字母表示标量。 x_i 是向量 $\mathbf{x}$ 的第 i^{th} 个元素，使用斜体是因为单个向量元素是标量。还需要约定向量 $\mathbf{x}$ 的方向。我们默认所有向量都是大小为 $n \times 1$ 的列向量：

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

对于多个标量值函数，也可以像处理参数那样，把它们组合成一个向量。令 $\mathbf{y} = \mathbf{f}(\mathbf{x})$ 是由 m 个标量值函数组成的向量，每个函数都接受一个长度为 $n = |\mathbf{x}|$ 的向量 $\mathbf{x}$ ；其中 $|\mathbf{x}|$ 表示 $\mathbf{x}$ 中元素的基数（数量）。与上一节一样， $\mathbf{f}$ 中的每个函数 f_i 都返回一个标量：

$$\begin{aligned} y_1 &= f_1(\mathbf{x}) \\ y_2 &= f_2(\mathbf{x}) \\ &\vdots \\ y_m &= f_m(\mathbf{x}) \end{aligned}$$

例如，上一节的 $f(x, y) = 3x^2y$ 和 $g(x, y) = 2x + y^8$ 可以写成：

$$\begin{aligned} y_1 &= f_1(\mathbf{x}) = 3x_1^2x_2 && (\text{substituting } x_1 \text{ for } x, x_2 \text{ for } y) \\ y_2 &= f_2(\mathbf{x}) = 2x_1 + x_2^8 \end{aligned}$$

常见的情形是 $m = n$ ，因为向量 $\mathbf{x}$ 中的每个元素都对应一个标量函数结果。例如，考虑恒等函数 $\mathbf{y} = \mathbf{f}(\mathbf{x}) = \mathbf{x}$ ：

$$\begin{aligned} y_1 &= f_1(\mathbf{x}) = x_1 \\ y_2 &= f_2(\mathbf{x}) = x_2 \\ &\vdots \\ y_n &= f_n(\mathbf{x}) = x_n \end{aligned}$$

在这个例子中，函数和参数的数量均为 $m = n$ 。一般而言，雅可比矩阵汇集了全部 $m \times n$ 个可能的偏导数（ m 行、 n 列），也就是将 m 个对 $\mathbf{x}$ 的梯度堆叠起来：

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \\ \vdots \\ \nabla f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}} f_1(\mathbf{x}) \\ \frac{\partial}{\partial \mathbf{x}} f_2(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial \mathbf{x}} f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(\mathbf{x}) & \frac{\partial}{\partial x_2} f_1(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n} f_1(\mathbf{x}) \\ \frac{\partial}{\partial x_1} f_2(\mathbf{x}) & \frac{\partial}{\partial x_2} f_2(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n} f_2(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} f_m(\mathbf{x}) & \frac{\partial}{\partial x_2} f_m(\mathbf{x}) & \cdots & \frac{\partial}{\partial x_n} f_m(\mathbf{x}) \end{bmatrix}$$

每个 $\frac{\partial}{\partial \mathbf{x}} f_i(\mathbf{x})$ 都是一个长度为 n 的行向量，因为偏导数所针对的向量 $\mathbf{x}$ 长度为 $n = |\mathbf{x}|$ 。对 $\mathbf{x}$ 求偏导时，雅可比矩阵的宽度是 n ，因为有 n 个可以改变的参数，每个都可能影响函数值。因此，对于 m 个方程，雅可比矩阵总有 m 行。将这些可能的形状画出来，有助于理解：

	vector	
	scalar	
	$\boxed{x}$	$\boxed{\mathbf{x}}$
scalar		
$\boxed{f}$	$\boxed{\frac{\partial f}{\partial x}}$	$\boxed{\frac{\partial f}{\partial \mathbf{x}}}$
vector		
$\boxed{\mathbf{f}}$	$\boxed{\frac{\partial \mathbf{f}}{\partial x}}$	$\boxed{\frac{\partial \mathbf{f}}{\partial \mathbf{x}}}$

恒等函数 $\mathbf{f}(\mathbf{x}) = \mathbf{x}$ 的雅可比矩阵 (其中 $f_i(\mathbf{x}) = x_i$) 涉及 n 个函数，每个函数都有 n 个参数，统一放在向量 $\mathbf{x}$ 中。因此，雅可比矩阵是方阵，因为 $m = n$ ：

$$\begin{aligned}
 \frac{\partial \mathbf{y}}{\partial \mathbf{x}} &= \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}} f_1(\mathbf{x}) \\ \frac{\partial}{\partial \mathbf{x}} f_2(\mathbf{x}) \\ \dots \\ \frac{\partial}{\partial \mathbf{x}} f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(\mathbf{x}) & \frac{\partial}{\partial x_2} f_1(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_1(\mathbf{x}) \\ \frac{\partial}{\partial x_1} f_2(\mathbf{x}) & \frac{\partial}{\partial x_2} f_2(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_2(\mathbf{x}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial}{\partial x_1} f_m(\mathbf{x}) & \frac{\partial}{\partial x_2} f_m(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_m(\mathbf{x}) \end{bmatrix} \\
 &= \begin{bmatrix} \frac{\partial}{\partial x_1} x_1 & \frac{\partial}{\partial x_2} x_1 & \dots & \frac{\partial}{\partial x_n} x_1 \\ \frac{\partial}{\partial x_1} x_2 & \frac{\partial}{\partial x_2} x_2 & \dots & \frac{\partial}{\partial x_n} x_2 \\ \dots & \dots & \dots & \dots \\ \frac{\partial}{\partial x_1} x_n & \frac{\partial}{\partial x_2} x_n & \dots & \frac{\partial}{\partial x_n} x_n \end{bmatrix} \\
 &\quad \text{(and since } \frac{\partial}{\partial x_j} x_i = 0 \text{ for } j \neq i) \\
 &= \begin{bmatrix} \frac{\partial}{\partial x_1} x_1 & 0 & \dots & 0 \\ 0 & \frac{\partial}{\partial x_2} x_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \frac{\partial}{\partial x_n} x_n \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix} \\
 &= \mathbf{I} \quad (\mathbf{I} \text{ is the identity matrix with ones down the diagonal})
 \end{aligned}$$

继续之前，请确保自己能推导上面的每一步。如果卡住了，就单独考虑矩阵中的每个元素，应用普通的标量求导法则。这是一个普遍有用的技巧：先把向量表达式拆成一组标量表达式，再计算所有偏导数，最后将结果适当地组合成向量和矩阵。

还要仔细区分矩阵是纵向的 $\mathbf{x}$ ，还是横向的 $\mathbf{x}^T$ ；其中 $\mathbf{x}^T$ 表示 $\mathbf{x}$ 的转置。同时，也要注意区分标量值函数 $y = \dots$ 与函数向量 (或向量值函数) $\mathbf{y} = \dots$ 。

向量逐元素二元运算的导数

向量的逐元素二元运算，例如向量加法 $\mathbf{w} + \mathbf{x}$ ，非常重要，因为许多常见向量运算（如向量乘以标量）都可以表示为逐元素二元运算。所谓“逐元素二元运算”，就是将算子作用于两个向量的第一个元素，得到输出的第一个元素；再作用于第二个元素，得到输出的第二个元素，依此类推。例如，NumPy 或 TensorFlow 默认就是这样应用基本数学运算符的。深度学习中常见的例子有 $\max(\mathbf{w}, \mathbf{x})$ 和 $\mathbf{w} > \mathbf{x}$ （后者返回由 1 和 0 组成的向量）。

可以用 $\mathbf{y} = \mathbf{f}(\mathbf{w}) \circ \mathbf{g}(\mathbf{x})$ 概括逐元素二元运算，其中 $m = n = |\mathbf{y}| = |\mathbf{w}| = |\mathbf{x}|$ 。（提醒一下： $|\mathbf{x}|$ 是 $\mathbf{x}$ 中元素的数量。）符号 $\circ$ 表示任意逐元素运算符（例如 $+$ ），而不是函数复合运算符 $\circ$ 。将方程 $\mathbf{y} = \mathbf{f}(\mathbf{w}) \circ \mathbf{g}(\mathbf{x})$ 展开到标量层面，就得到：

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{w}) \circ g_1(\mathbf{x}) \\ f_2(\mathbf{w}) \circ g_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{w}) \circ g_n(\mathbf{x}) \end{bmatrix}$$

这里将 n 个（而不是 m 个）方程竖着排列，是为了强调逐元素运算的结果是大小为 $m = n$ 的向量。

根据上一节的思路可知，对 $\mathbf{w}$ 的雅可比矩阵在一般情况下是如下方阵：

$$J_{\mathbf{w}} = \frac{\partial \mathbf{y}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial}{\partial w_1}(f_1(\mathbf{w}) \circ g_1(\mathbf{x})) & \frac{\partial}{\partial w_2}(f_1(\mathbf{w}) \circ g_1(\mathbf{x})) & \cdots & \frac{\partial}{\partial w_n}(f_1(\mathbf{w}) \circ g_1(\mathbf{x})) \\ \frac{\partial}{\partial w_1}(f_2(\mathbf{w}) \circ g_2(\mathbf{x})) & \frac{\partial}{\partial w_2}(f_2(\mathbf{w}) \circ g_2(\mathbf{x})) & \cdots & \frac{\partial}{\partial w_n}(f_2(\mathbf{w}) \circ g_2(\mathbf{x})) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial w_1}(f_n(\mathbf{w}) \circ g_n(\mathbf{x})) & \frac{\partial}{\partial w_2}(f_n(\mathbf{w}) \circ g_n(\mathbf{x})) & \cdots & \frac{\partial}{\partial w_n}(f_n(\mathbf{w}) \circ g_n(\mathbf{x})) \end{bmatrix}$$

而对 $\mathbf{x}$ 的雅可比矩阵为：

$$J_{\mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial}{\partial x_1}(f_1(\mathbf{w}) \circ g_1(\mathbf{x})) & \frac{\partial}{\partial x_2}(f_1(\mathbf{w}) \circ g_1(\mathbf{x})) & \cdots & \frac{\partial}{\partial x_n}(f_1(\mathbf{w}) \circ g_1(\mathbf{x})) \\ \frac{\partial}{\partial x_1}(f_2(\mathbf{w}) \circ g_2(\mathbf{x})) & \frac{\partial}{\partial x_2}(f_2(\mathbf{w}) \circ g_2(\mathbf{x})) & \cdots & \frac{\partial}{\partial x_n}(f_2(\mathbf{w}) \circ g_2(\mathbf{x})) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1}(f_n(\mathbf{w}) \circ g_n(\mathbf{x})) & \frac{\partial}{\partial x_2}(f_n(\mathbf{w}) \circ g_n(\mathbf{x})) & \cdots & \frac{\partial}{\partial x_n}(f_n(\mathbf{w}) \circ g_n(\mathbf{x})) \end{bmatrix}$$

看上去相当复杂，但幸运的是，雅可比矩阵常常是对角矩阵，也就是除了主对角线之外，其他位置全为零的矩阵。这会极大地简化计算，因此我们仔细看看：对于逐元素运算，雅可比矩阵何时会化为对角矩阵？

在对角雅可比矩阵中，所有非对角元素都为零： $\frac{\partial}{\partial w_j}(f_i(\mathbf{w}) \circ g_i(\mathbf{x})) = 0$ ，其中 $j \neq i$ 。（注意，这里是对 w_j 而不是 w_i 求偏导。）这些非对角元素在什么条件下为零？正是在 f_i 和 g_i 对 w_j 而言都是常数时，即 $\frac{\partial}{\partial w_j} f_i(\mathbf{w}) = \frac{\partial}{\partial w_j} g_i(\mathbf{x}) = 0$ 。无论使用什么运算符，只要这些偏导数为零，整个运算结果对该变量的导数就为零，即 $0 \circ 0 = 0$ ，因为常数的偏导数是零。

当 f_i 和 g_i 不依赖于 w_j 时，这些偏导数就为零。我们知道，逐元素运算意味着 f_i 仅是 w_i 的函数， g_i 仅是 x_i 的函数。例如， $\mathbf{w} + \mathbf{x}$ 计算的是 $w_i + x_i$ 之和。因此， $f_i(\mathbf{w}) \circ g_i(\mathbf{x})$ 化为 $f_i(w_i) \circ g_i(x_i)$ ，我们的目标也变成 $\frac{\partial}{\partial w_j} f_i(w_i) = \frac{\partial}{\partial w_j} g_i(x_i) = 0$ 。当 $j \neq i$ 时，对 w_j 求偏导的算子会将 $f_i(w_i)$ 和 $g_i(x_i)$ 视为常数，所以非对角位置的偏导数为零。（严格来说， $f_i(w_i)$ 是对记号的一种滥用，因为 f_i 和 g_i 的参数是向量，而不是单个元素。我们本应写成类似 $\hat{f}_i(w_i) = f_i(\mathbf{w})$ 的形式，但这会让式子更加混乱；程序员通常也习惯函数重载，因此下文仍沿用这一记法。）

后面我们会利用这种简化，并将“ $f_i(\mathbf{w})$ 和 $g_i(\mathbf{x})$ 分别最多只访问 w_i 和 x_i ”这一约束称为逐元素对角条件。

在这一条件下，雅可比矩阵的对角元素为 $\frac{\partial}{\partial w_i}(f_i(w_i) \circ g_i(x_i))$ ：

$$\frac{\partial \mathbf{y}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial}{\partial w_1}(f_1(w_1) \circ g_1(x_1)) & & & 0 \\ & \frac{\partial}{\partial w_2}(f_2(w_2) \circ g_2(x_2)) & & \\ & & \dots & \\ 0 & & & \frac{\partial}{\partial w_n}(f_n(w_n) \circ g_n(x_n)) \end{bmatrix}$$

(大号的“0”是一种简写，表示所有非对角元素都为 0。)

更简洁地，可以写成：

$$\frac{\partial \mathbf{y}}{\partial \mathbf{w}} = \text{diag} \left(\frac{\partial}{\partial w_1}(f_1(w_1) \circ g_1(x_1)), \frac{\partial}{\partial w_2}(f_2(w_2) \circ g_2(x_2)), \dots, \frac{\partial}{\partial w_n}(f_n(w_n) \circ g_n(x_n)) \right)$$

以及

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \text{diag} \left(\frac{\partial}{\partial x_1}(f_1(w_1) \circ g_1(x_1)), \frac{\partial}{\partial x_2}(f_2(w_2) \circ g_2(x_2)), \dots, \frac{\partial}{\partial x_n}(f_n(w_n) \circ g_n(x_n)) \right)$$

其中， $\text{diag}(\mathbf{x})$ 构造一个对角元素取自向量 $\mathbf{x}$ 的矩阵。

由于经常进行简单的向量算术，逐元素二元运算中的一般函数 $f(\mathbf{w})$ 往往就是向量 $\mathbf{w}$ 本身。只要这个一般函数就是向量本身， $f_i(\mathbf{w})$ 就化为 $f_i(w_i) = w_i$ 。例如，向量加法 $\mathbf{w} + \mathbf{x}$ 满足逐元素对角条件，因为 $f(\mathbf{w}) + g(\mathbf{x})$ 对应的标量方程 $y_i = f_i(\mathbf{w}) + g_i(\mathbf{x})$ 可以简化为 $y_i = f_i(w_i) + g_i(x_i) = w_i + x_i$ ，其偏导数为：

$$\frac{\partial}{\partial w_i}(f_i(w_i) + g_i(x_i)) = \frac{\partial}{\partial w_i}(w_i + x_i) = 1 + 0 = 1$$

$$\frac{\partial}{\partial x_i}(f_i(w_i) + g_i(x_i)) = \frac{\partial}{\partial x_i}(w_i + x_i) = 0 + 1 = 1$$

由此得到 $\frac{\partial(\mathbf{w}+\mathbf{x})}{\partial \mathbf{w}} = \frac{\partial(\mathbf{w}+\mathbf{x})}{\partial \mathbf{x}} = I$ ，也就是单位矩阵，因为每个对角元素都为 1。 I 表示维度适当的方形单位矩阵：对角线全为 1，其余位置全为零。

由于 $f_i(\mathbf{w})$ 化为 $f_i(w_i)$ 这一特例十分简单，你应该能够自行推导常见向量逐元素二元运算的雅可比矩阵：

Op	Partial with respect to $\mathbf{w}$
+	$\frac{\partial(\mathbf{w}+\mathbf{x})}{\partial \mathbf{w}} = \text{diag}(\dots \frac{\partial(w_i+x_i)}{\partial w_i} \dots) = \text{diag}(\vec{1}) = I$
-	$\frac{\partial(\mathbf{w}-\mathbf{x})}{\partial \mathbf{w}} = \text{diag}(\dots \frac{\partial(w_i-x_i)}{\partial w_i} \dots) = \text{diag}(\vec{1}) = I$
$\otimes$	$\frac{\partial(\mathbf{w} \otimes \mathbf{x})}{\partial \mathbf{w}} = \text{diag}(\dots \frac{\partial(w_i \times x_i)}{\partial w_i} \dots) = \text{diag}(\mathbf{x})$
$\odot$	$\frac{\partial(\mathbf{w} \odot \mathbf{x})}{\partial \mathbf{w}} = \text{diag}(\dots \frac{\partial(w_i/x_i)}{\partial w_i} \dots) = \text{diag}(\dots \frac{1}{x_i} \dots)$

$$\begin{aligned}
&\text{Op} && \text{Partial with respect to } \mathbf{x} \\
+ &&& \frac{\partial(\mathbf{w}+\mathbf{x})}{\partial \mathbf{x}} = \mathbf{I} \\
- &&& \frac{\partial(\mathbf{w}-\mathbf{x})}{\partial \mathbf{x}} = \text{diag}(\dots \frac{\partial(w_i-x_i)}{\partial x_i} \dots) = \text{diag}(-\vec{1}) = -\mathbf{I} \\
\otimes &&& \frac{\partial(\mathbf{w} \otimes \mathbf{x})}{\partial \mathbf{x}} = \text{diag}(\mathbf{w}) \\
\odot &&& \frac{\partial(\mathbf{w} \odot \mathbf{x})}{\partial \mathbf{x}} = \text{diag}(\dots \frac{-w_i}{x_i^2} \dots)
\end{aligned}$$

算子 $\otimes$ 和 $\odot$ 分别表示逐元素乘法和除法； $\otimes$ 有时称为哈达玛积 (Hadamard product)。逐元素乘除没有统一的标准记号，所以这里采用与一般二元运算记法一致的方式。

涉及标量扩展的导数

将标量与向量相乘或相加时，我们隐式地把标量扩展为向量，再执行逐元素二元运算。例如，标量 z 加到向量 $\mathbf{x}$ 上得到 $\mathbf{y} = \mathbf{x} + z$ ，实际就是 $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(z)$ ，其中 $\mathbf{f}(\mathbf{x}) = \mathbf{x}$ 且 $\mathbf{g}(z) = \vec{1}z$ 。（记号 $\vec{1}$ 表示长度适当的全 1 向量。） z 是任意不依赖于 $\mathbf{x}$ 的标量；这一点很有用，因为此时对任意 x_i 都有 $\frac{\partial z}{\partial x_i} = 0$ ，从而简化偏导数计算。（在这里的讨论中，把变量 z 当作常数就可以。）类似地，标量乘法 $\mathbf{y} = \mathbf{x}z$ 实际上是 $\mathbf{y} = \mathbf{f}(\mathbf{x}) \otimes \mathbf{g}(z) = \mathbf{x} \otimes \vec{1}z$ ，其中 $\otimes$ 表示两个向量的逐元素乘法（哈达玛积）。

对于向量与标量的加法、乘法，对向量 $\mathbf{x}$ 的偏导数可以用逐元素法则计算：

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \text{diag} \left(\dots \frac{\partial}{\partial x_i} (f_i(x_i) \odot g_i(z)) \dots \right)$$

这是因为函数 $\mathbf{f}(\mathbf{x}) = \mathbf{x}$ 和 $\mathbf{g}(z) = \vec{1}z$ 显然满足雅可比矩阵的逐元素对角条件： $f_i(\mathbf{x})$ 最多只涉及 x_i ，而 $g_i(z)$ 只涉及向量 $\vec{1}z$ 的第 i^{th} 个值。

利用通常的标量偏导数法则，可以得到向量与标量相加时雅可比矩阵的对角元素：

$$\frac{\partial}{\partial x_i} (f_i(x_i) + g_i(z)) = \frac{\partial(x_i + z)}{\partial x_i} = \frac{\partial x_i}{\partial x_i} + \frac{\partial z}{\partial x_i} = 1 + 0 = 1$$

所以， $\frac{\partial}{\partial \mathbf{x}}(\mathbf{x} + z) = \text{diag}(\vec{1}) = \mathbf{I}$ 。

不过，对标量参数 z 求偏导，得到的是一个列向量，而不是对角矩阵。该向量的元素为：

$$\frac{\partial}{\partial z} (f_i(x_i) + g_i(z)) = \frac{\partial(x_i + z)}{\partial z} = \frac{\partial x_i}{\partial z} + \frac{\partial z}{\partial z} = 0 + 1 = 1$$

因此， $\frac{\partial}{\partial z}(\mathbf{x} + z) = \vec{1}$ 。

向量与标量相乘时，雅可比矩阵的对角元素需要用到标量求导的乘积法则：

$$\frac{\partial}{\partial x_i} (f_i(x_i) \otimes g_i(z)) = x_i \frac{\partial z}{\partial x_i} + z \frac{\partial x_i}{\partial x_i} = 0 + z = z$$

所以， $\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}z) = \text{diag}(\vec{1}z) = \mathbf{I}z$ 。

对标量参数 z 的偏导数是一个列向量，其元素为：

$$\frac{\partial}{\partial z}(f_i(x_i) \otimes g_i(z)) = x_i \frac{\partial z}{\partial z} + z \frac{\partial x_i}{\partial z} = x_i + 0 = x_i$$

由此得到 $\frac{\partial}{\partial z}(\mathbf{x}z) = \mathbf{x}$ 。

向量求和归约

将向量元素求和是深度学习中的重要操作，例如网络损失函数中就会用到。它也能帮助我们简化向量点积及其他将向量归约为标量的运算的求导。

令 $y = \text{sum}(\mathbf{f}(\mathbf{x})) = \sum_{i=1}^n f_i(\mathbf{x})$ 。注意，这里特意将参数保留为向量 $\mathbf{x}$ ，因为每个函数 f_i 都可能使用向量中的全部值，而不只是 x_i 。求和针对的是函数的结果，而不是参数。向量求和的梯度（大小为 $1 \times n$ 的雅可比矩阵）为：

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{x}} &= \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n} \right] \\ &= \left[\frac{\partial}{\partial x_1} \sum_i f_i(\mathbf{x}), \frac{\partial}{\partial x_2} \sum_i f_i(\mathbf{x}), \dots, \frac{\partial}{\partial x_n} \sum_i f_i(\mathbf{x}) \right] \\ &= \left[\sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_1}, \sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_2}, \dots, \sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_n} \right] \quad (\text{move derivative inside } \sum) \end{aligned}$$

（梯度各分量内部的求和容易混淆，因此一定要保持符号一致。）

来看看简单函数 $y = \text{sum}(\mathbf{x})$ 的梯度。求和内部的函数就是 $f_i(\mathbf{x}) = x_i$ ，于是梯度为：

$$\nabla y = \left[\sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_1}, \sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_2}, \dots, \sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_n} \right] = \left[\sum_i \frac{\partial x_i}{\partial x_1}, \sum_i \frac{\partial x_i}{\partial x_2}, \dots, \sum_i \frac{\partial x_i}{\partial x_n} \right]$$

由于在 $j \neq i$ 时有 $\frac{\partial}{\partial x_j} x_i = 0$ ，可以化简为：

$$\nabla y = \left[\frac{\partial x_1}{\partial x_1}, \frac{\partial x_2}{\partial x_2}, \dots, \frac{\partial x_n}{\partial x_n} \right] = [1, 1, \dots, 1] = \vec{1}^T$$

注意，结果是全 1 的行向量，而非列向量，所以梯度为 $\vec{1}^T$ 。（ $\vec{1}^T$ 中的上标 T 表示该向量的转置，在这里就是将列向量变成行向量。）务必始终理清所有向量和矩阵的形状，否则就无法计算复杂函数的导数。

再看一个例子：向量乘以常数标量之后，对结果求和。若 $y = \text{sum}(\mathbf{x}z)$ ，则 $f_i(\mathbf{x}, z) = x_i z$ 。梯度为：

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{x}} &= \left[\sum_i \frac{\partial}{\partial x_1} x_i z, \sum_i \frac{\partial}{\partial x_2} x_i z, \dots, \sum_i \frac{\partial}{\partial x_n} x_i z \right] \\ &= \left[\frac{\partial}{\partial x_1} x_1 z, \frac{\partial}{\partial x_2} x_2 z, \dots, \frac{\partial}{\partial x_n} x_n z \right] \\ &= [z, z, \dots, z] \end{aligned}$$

对标量变量 z 的导数为 1×1 ：

$$\begin{aligned}
\frac{\partial y}{\partial z} &= \frac{\partial}{\partial z} \sum_{i=1}^n x_i z \\
&= \sum_i \frac{\partial}{\partial z} x_i z \\
&= \sum_i x_i \\
&= \text{sum}(\mathbf{x})
\end{aligned}$$

链式法则

仅靠目前介绍的基本矩阵微积分法则，还无法计算非常复杂函数的偏导数。例如，对于 $\text{sum}(\mathbf{w} + \mathbf{x})$ 这样的嵌套表达式，若不先将其展开为等价的标量形式，就无法直接求导。我们需要一种能将基本向量法则组合起来的方法，可以称之为向量链式法则。遗憾的是，有多种求导法则都叫“链式法则”，因此必须明确我们说的是哪一种。本节的目标之一，就是清楚地定义并命名三种不同的链式法则，说明它们各自适用的情形。先从单变量链式法则开始：求标量函数对标量的导数。然后介绍全导数这一重要概念，并据此定义一个名称略显冗长的法则：单变量全导数链式法则。最后，我们就能完整地讨论神经网络所需的向量链式法则。

从概念上说，链式法则是一种分治策略（类似快速排序）：将复杂表达式拆成导数更容易计算的子表达式。它的威力在于，我们可以分别处理每个简单子表达式，再把中间结果组合起来，依然得到正确的整体结果。

当我们需要对包含嵌套子表达式的表达式求导时，链式法则就派上用场了。例如，处理 $\frac{d}{dx} \sin(x^2)$ 就需要链式法则。最外层对中间结果求 $\sin$ ，而内层子表达式则对 x 求平方。具体来说，这里需要单变量链式法则，下面详细讨论。

单变量链式法则

先给出这个嵌套表达式的导数： $\frac{d}{dx} \sin(x^2) = 2x \cos(x^2)$ 。不需要是数学天才，也能看出结果中有两个熟悉的标量求导成分： $\frac{d}{dx} x^2 = 2x$ 和 $\frac{d}{du} \sin(u) = \cos(u)$ 。看来，只要把外层表达式的导数乘以内层表达式的导数，也就是“把各部分串起来”，就能得到答案。确实如此。本节将说明其一般原理，并给出适用于单变量深层嵌套表达式的计算流程。

链式法则通常用嵌套函数来定义，例如单变量链式法则中的 $y = f(g(x))$ 。（也会看到用函数复合 $(f \circ g)(x)$ 定义的形式，含义相同。）有些资料将导数简写为 $y' = f'(g(x))g'(x)$ ，但这掩盖了我们引入中间变量 $u = g(x)$ 的事实，稍后会看到这一点。最好明确写出 $f(g(x))$ 的**单变量链式法则**，以免对错误的变量求导。我们推荐以下形式：

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

使用单变量链式法则时，遵循以下步骤：

1. 为嵌套子表达式，以及二元和一元运算的子表达式引入中间变量。例如， $\times$ 是二元运算； $\sin(x)$ 和其他三角函数通常是一元运算，因为只有一个操作数。这一步将所有方程规范化为单个运算或单次函数调用。
2. 计算各中间变量对其参数的导数。
3. 将各中间变量的导数相乘，得到整体结果。

4. 如果最终导数表达式中仍引用了中间变量，就代回它们的原始表达式。

第三步把中间结果串成一条“链”，这正是“链式法则”名称的由来。将中间导数相乘，是各种链式法则共有的主题。

用这个流程处理 $y = f(g(x)) = \sin(x^2)$ ：

1. 引入中间变量。令 $u = x^2$ 表示子表达式 x^2 (是以下表达式的简写： $u(x) = x^2$)。由此得到：

$$\begin{aligned} u &= x^2 && \text{(relative to definition } f(g(x)), g(x) = x^2) \\ y &= \sin(u) && (y = f(u) = \sin(u)) \end{aligned}$$

这些子表达式的排列顺序不影响答案，但我们建议按嵌套结构由内向外展开。这样，每个表达式及其导数都只依赖之前已经计算出的量。

2. 计算导数。

$$\begin{aligned} \frac{du}{dx} &= 2x && \text{(Take derivative with respect to } x) \\ \frac{dy}{du} &= \cos(u) && \text{(Take derivative with respect to } u \text{ not } x) \end{aligned}$$

3. 组合。

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx} = \cos(u)2x$$

4. 代回。

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx} = \cos(x^2)2x = 2x\cos(x^2)$$

注意，分别计算各个中间变量的导数有多么简单！链式法则保证这种做法成立，并告诉我们如何组合中间结果，得到 $2x\cos(x^2)$ 。

可以用单位约消来理解链式法则的组合步骤。设 y 的单位是英里， x 表示油箱中的加仑数， u 的单位也是加仑，那么 $\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$ 可以理解为 $\frac{\text{miles}}{\text{tank}} = \frac{\text{miles}}{\text{gallon}} \frac{\text{gallon}}{\text{tank}}$ 。分母和分子中的 *gallon* 会相互约消。

理解单变量链式法则的另一种方式，是把整个表达式画成数据流图或运算链（对编译器开发者而言，就是抽象语法树）：

$$\begin{array}{c} y = \sin \\ \uparrow \\ u = \text{square} \\ \uparrow \\ x \end{array} \left\{ \begin{array}{l} \frac{dy}{du} \\ \frac{du}{dx} \end{array} \right.$$

函数参数 x 的变化先经过平方运算，再经过 $\sin$ 运算，最终改变结果 y 。可以将 $\frac{du}{dx}$ 理解为“把变化从 x 传到 u ”，将 $\frac{dy}{du}$ 理解为“把变化从 u 传到 y ”。从 x 到 y 需要经过一个中间环节。按照惯例，链式法则通常从输出变量向参数方向书写，即 $\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$ 。但若从 x 到 y 的角度看，反过来写成等价的 $\frac{dy}{dx} = \frac{du}{dx} \frac{dy}{du}$ 会更直观。

单变量链式法则的适用条件。注意，从 x 到根节点 y 只有一条数据流路径。 x 的变化只能通过一种方式影响输出 y ，这就是应用单变量链式法则的条件。另一个更容易记住、但不那么严密的说法是：所有中间子表达式函

数（如 $u(x)$ 和 $y(u)$ ）都不超过一个参数。考虑 $y(x) = x + x^2$ ，引入中间变量 u 后，它变成 $y(x, u) = x + u$ 。下一节会看到，在 $y(x, u)$ 中，从 x 到 y 有多条路径。处理这种情况需要单变量全导数链式法则。

给对自动微分感兴趣的读者补充一点：论文和库文档会使用前向微分与反向微分（用于反向传播算法）这些术语。从数据流的角度看，我们正在做前向微分，因为它沿着正常的数据流方向进行。反向微分则沿相反方向，追问输出的变化如何关联到函数参数 x 。由于反向微分能同时确定所有函数参数对应的变化，对于参数很多的函数，其导数计算会高效得多。前向微分则需要依次考虑每个参数的变化如何影响函数输出 y 。下表着重展示这两种方法计算偏导数的顺序。

从 x 到 y 的前向微分	从 y 到 x 的反向微分
$\frac{dy}{dx} = \frac{du}{dx} \frac{dy}{du}$	$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$

自动微分超出了本文的范围，这里只是为今后的文章作铺垫。

许多读者可以在脑中算出 $\frac{d}{dx} \sin(x^2)$ ，但我们的目标是建立一套即使面对非常复杂的表达式也能使用的流程。这也正是 PyTorch 等库中[自动微分](#)的工作方式。因此，按这种方式手工求导，也是在学习如何为 PyTorch 中的自定义神经网络定义函数。

面对深层嵌套表达式，可以把使用链式法则想象成编译器将 $f_4(f_3(f_2(f_1(x))))$ 这样的嵌套函数调用，展开为一串调用的过程。函数 f_i 的返回结果被存入一个称为寄存器的临时变量，再作为参数传给 f_{i+1} 。下面把我们的流程应用于 $y = f(x) = \ln(\sin(x^3)^2)$ 这样的深层嵌套方程，看看实际效果：

1. 引入中间变量。

$$\begin{aligned} u_1 &= f_1(x) = x^3 \\ u_2 &= f_2(u_1) = \sin(u_1) \\ u_3 &= f_3(u_2) = u_2^2 \\ u_4 &= f_4(u_3) = \ln(u_3) \quad (y = u_4) \end{aligned}$$

2. 计算导数。

$$\begin{aligned} \frac{d}{dx} u_1 &= \frac{d}{dx} x^3 = 3x^2 \\ \frac{d}{du_1} u_2 &= \frac{d}{du_1} \sin(u_1) = \cos(u_1) \\ \frac{d}{du_2} u_3 &= \frac{d}{du_2} u_2^2 = 2u_2 \\ \frac{d}{du_3} u_4 &= \frac{d}{du_3} \ln(u_3) = \frac{1}{u_3} \end{aligned}$$

3. 组合四个中间值。

$$\frac{dy}{dx} = \frac{du_4}{dx} = \frac{du_4}{du_3} \frac{du_3}{du_2} \frac{du_2}{du_1} \frac{du_1}{dx} = \frac{1}{u_3} 2u_2 \cos(u_1) 3x^2 = \frac{6u_2 x^2 \cos(u_1)}{u_3}$$

4. 代回。

$$\frac{dy}{dx} = \frac{6\sin(u_1)x^2\cos(x^3)}{u_2^2} = \frac{6\sin(x^3)x^2\cos(x^3)}{\sin(u_1)^2} = \frac{6\sin(x^3)x^2\cos(x^3)}{\sin(x^3)^2} = \frac{6x^2\cos(x^3)}{\sin(x^3)}$$

下面展示从 x 到 y 的运算链中的数据流：

$$\begin{array}{c}
 y = r_4 \\
 \uparrow \\
 r_3 \text{ square} \\
 \uparrow \\
 r_2 \text{ sin} \\
 \uparrow \\
 r_1 \text{ cube} \\
 \uparrow \\
 x
 \end{array}
 \left\{
 \begin{array}{l}
 \frac{du_4}{du_3} \\
 \frac{du_3}{du_2} \\
 \frac{du_2}{du_1} \\
 \frac{du_1}{dx}
 \end{array}
 \right.$$

至此，我们可以用链式法则处理单变量 x 的嵌套表达式导数，但前提是 x 只能通过一条数据流路径影响 y 。为了处理更复杂的表达式，接下来需要扩展这一方法。

单变量全导数链式法则

单变量链式法则的适用范围有限，因为所有中间变量都必须是单变量函数。不过，它展示了链式法则的核心机制：将中间子表达式的导数相乘。要处理 $y = f(x) = x + x^2$ 这样的更一般表达式，我们需要扩展这条基本法则。

当然，我们一眼就能看出 $\frac{dy}{dx} = \frac{d}{dx}x + \frac{d}{dx}x^2 = 1 + 2x$ ，但这用的是标量加法求导法则，而非链式法则。如果尝试直接应用单变量链式法则，就会得到错误答案。事实上，前面的链式法则在这里并不适用，因为求导算子 $\frac{d}{dx}$ 不适用于中间变量中的多元函数，例如 u_2 ：

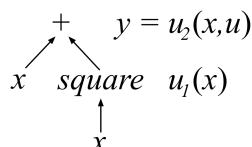
$$\begin{aligned}
 u_1(x) &= x^2 \\
 u_2(x, u_1) &= x + u_1 \quad (y = f(x) = u_2(x, u_1))
 \end{aligned}$$

不妨硬试一下，看看会发生什么。如果假装 $\frac{du_2}{du_1} = 0 + 1 = 1$ 且 $\frac{du_1}{dx} = 2x$ ，那么得到的是 $\frac{dy}{dx} = \frac{du_2}{dx} = \frac{du_2}{du_1} \frac{du_1}{dx} = 2x$ ，而不是正确答案 $1 + 2x$ 。

由于 $u_2(x, u) = x + u_1$ 有多个参数，偏导数就登场了。我们先不加辨别地对所有方程使用偏导算子，看看结果：

$$\begin{aligned}
 \frac{\partial u_1(x)}{\partial x} &= 2x && \text{(same as } \frac{du_1(x)}{dx} \text{)} \\
 \frac{\partial u_2(x, u_1)}{\partial u_1} &= \frac{\partial}{\partial u_1}(x + u_1) = 0 + 1 = 1 \\
 \frac{\partial u_2(x, u_1)}{\partial x} &\neq \frac{\partial}{\partial x}(x + u_1) = 1 + 0 = 1 && \text{(something's not quite right here!)}
 \end{aligned}$$

糟糕！偏导数 $\frac{\partial u_2(x, u_1)}{\partial x}$ 在这里给出了错误结果，因为这种用法违反了偏导数的一个关键假设：对 x 求偏导时，其他变量不能随着 x 一起变化，否则就不能把它们当作常数。然而， $u_1(x) = x^2$ 显然是 x 的函数，因此会随 x 变化。由于 $\frac{\partial u_1(x)}{\partial x} \neq 0$ ，所以 $\frac{\partial u_2(x, u_1)}{\partial x} \neq 1 + 0$ 。看看 $y = u_2(x, u_1)$ 的数据流图，就会发现从 x 到 y 有多条路径，这表明我们必须同时考虑对 x 的直接依赖，以及通过 $u_1(x)$ 产生的间接依赖：



x 既作为加法的操作数，也作为平方运算的操作数影响 y 。下面的方程描述对 x 作微小调整如何影响输出：

$$\hat{y} = (x + \Delta x) + (x + \Delta x)^2$$

于是有 $\Delta y = \hat{y} - y$ ，可读作：“ y 的变化，是原来的 y 与调整 x 后的 y 之间的差。”

如果令 $x = 1$ ，则 $y = 1 + 1^2 = 2$ 。将 x 增加 1，即 $\Delta x = 1$ ，则有 $\hat{y} = (1 + 1) + (1 + 1)^2 = 2 + 4 = 6$ 。 y 的变化不是 $\partial u_2 / u_1$ 可能让我们误以为的 1，而是 $6 - 2 = 4$ ！

这时就需要全导数“法则”了。其基本含义是：计算 $\frac{dy}{dx}$ 时，必须将 x 的变化对 y 的变化产生的所有可能贡献加起来。对 x 求全导数时，假设所有变量（如本例中的 u_1 ）都是 x 的函数，因此可能随 x 改变。若 $f(x) = u_2(x, u_1)$ 既直接依赖 x ，又通过中间变量 $u_1(x)$ 间接依赖它，则其全导数为：

$$\frac{dy}{dx} = \frac{\partial f(x)}{\partial x} = \frac{\partial u_2(x, u_1)}{\partial x} = \frac{\partial u_2}{\partial x} \frac{\partial x}{\partial x} + \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} = \frac{\partial u_2}{\partial x} + \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x}$$

使用这个公式，就得到正确答案：

$$\frac{dy}{dx} = \frac{\partial f(x)}{\partial x} = \frac{\partial u_2}{\partial x} + \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} = 1 + 1 \times 2x = 1 + 2x$$

这就是我们所称的单变量全导数链式法则的一次应用：

$$\frac{\partial f(x, u_1, \dots, u_n)}{\partial x} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial u_1} \frac{\partial u_1}{\partial x} + \frac{\partial f}{\partial u_2} \frac{\partial u_2}{\partial x} + \dots + \frac{\partial f}{\partial u_n} \frac{\partial u_n}{\partial x} = \frac{\partial f}{\partial x} + \sum_{i=1}^n \frac{\partial f}{\partial u_i} \frac{\partial u_i}{\partial x}$$

全导数假设所有变量都可能存在相互依赖；偏导数则将除 x 之外的所有变量视为常数。

这里的符号有一个微妙之处。所有导数都写成偏导数，因为 f 和 u_i 是多变量函数。这与 [MathWorld](#) 的记法一致，但不同于 [Wikipedia](#)，后者使用 $df(x, u_1, \dots, u_n)/dx$ （也许是为了强调等式的全导数性质）。我们将继续使用偏导数记号，以便与下一节讨论的向量链式法则保持一致。

实际使用时，只要记住：对 x 求全导数时，其他变量也可能是 x 的函数，所以它们的贡献也要加进来。等式左侧看起来像普通偏导数，右侧实际上计算的是全导数。不过，许多临时变量通常只依赖一个参数，这时单变量全导数链式法则就退化为单变量链式法则。

考虑一个嵌套子表达式，例如 $f(x) = \sin(x + x^2)$ 。引入三个中间变量：

$$\begin{aligned} u_1(x) &= x^2 \\ u_2(x, u_1) &= x + u_1 \\ u_3(u_2) &= \sin(u_2) \quad (y = f(x) = u_3(u_2)) \end{aligned}$$

以及相应偏导数：

$$\begin{aligned} \frac{\partial u_1}{\partial x} &= 2x \\ \frac{\partial u_2}{\partial x} &= \frac{\partial x}{\partial x} + \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} = 1 + 1 \times 2x = 1 + 2x \\ \frac{\partial f(x)}{\partial x} &= \frac{\partial u_3}{\partial x} + \frac{\partial u_3}{\partial u_2} \frac{\partial u_2}{\partial x} = 0 + \cos(u_2) \frac{\partial u_2}{\partial x} = \cos(x + x^2)(1 + 2x) \end{aligned}$$

其中， $\frac{\partial u_2}{\partial x}$ 和 $\frac{\partial f(x)}{\partial x}$ 都包含 $\frac{\partial u_1}{\partial x}$ 项，用来计入全导数的贡献。

还要注意，全导数公式始终对 $\frac{\partial f}{\partial u_i} \frac{\partial u_i}{\partial x}$ 这些项求和，而不是相乘。你可能以为求和是因为原式（例如 $y = x + x^2$ ）本身把两项相加，其实不是。全导数之所以相加，是因为它表示所有 x 对 y 变化的贡献的加权和。例如，即使使用 $y = x \times x^2$ 代替 $y = x + x^2$ ，全导数链式法则仍然把各偏导项相加。（ $x \times x^2$ 可以化简为 x^3 ，但为了演示，这里不合并这些项。）中间变量和偏导数如下：

$$\begin{aligned}
u_1(x) &= x^2 \\
u_2(x, u_1) &= xu_1 \quad (y = f(x) = u_2(x, u_1)) \\
\frac{\partial u_1}{\partial x} &= 2x \\
\frac{\partial u_2}{\partial x} &= u_1 \quad (\text{for } u_2 = x + u_1, \frac{\partial u_2}{\partial x} = 1) \\
\frac{\partial u_2}{\partial u_1} &= x \quad (\text{for } u_2 = x + u_1, \frac{\partial u_2}{\partial u_1} = 1)
\end{aligned}$$

而全导数的形式仍然相同：

$$\frac{dy}{dx} = \frac{\partial u_2}{\partial x} + \frac{\partial u_2}{\partial u_1} \frac{du_1}{dx} = u_1 + x2x = x^2 + 2x^2 = 3x^2$$

中间变量所用的运算符改变时，变化的是偏导数（权重），而不是公式本身。

微积分基础较好的读者可能会疑惑：为什么连 $x + x^2$ 中的 x^2 这种非嵌套子表达式，也要积极地引入中间变量？原因有三点：(i) 简化后的子表达式通常很容易求导；(ii) 这样可以简化链式法则；(iii) 这一流程与神经网络库中的自动微分机制一致。

进一步使用中间变量，就可以把单变量全导数链式法则化为最终形式。目标是去掉前面那个显得突兀的 $\frac{\partial f}{\partial x}$ ：

$$\frac{\partial f(x, u_1, \dots, u_n)}{\partial x} = \frac{\partial f}{\partial x} + \sum_{i=1}^n \frac{\partial f}{\partial u_i} \frac{\partial u_i}{\partial x}$$

只需引入一个新的临时变量，作为 x 的别名： $u_{n+1} = x$ ，就能做到。于是公式化为最终形式：

$$\frac{\partial f(u_1, \dots, u_{n+1})}{\partial x} = \sum_{i=1}^{n+1} \frac{\partial f}{\partial u_i} \frac{\partial u_i}{\partial x}$$

当所有中间变量都是单变量函数时，这条全导数链式法则就退化为单变量链式法则。因此，只记住这个更一般的公式，就能覆盖两种情况。先埋个伏笔：注意，这个求和很像向量点积 $\frac{\partial f}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial x}$ ，或者向量乘法 $\frac{\partial f}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial x}$ 。

继续之前，提醒一下网上的术语用法。遗憾的是，在微积分讨论中，本节这条基于全导数的链式法则通常被称为“多变量链式法则”，这个名称很容易造成误解！只有中间变量才是多元函数；整个函数（如 $f(x) = x + x^2$ ）仍是只接受一个参数 x 的标量函数。导数和参数都是标量，而不是“多元链式法则”这个名字可能让人联想到的向量。（在不涉及矩阵微积分的课堂上，“多元链式法则”大概不会造成歧义。）为了减少混淆，我们用“单变量全导数链式法则”这个名称，明确区分它与简单的单变量链式法则 $\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$ 。

向量链式法则

理解全导数链式法则之后，就可以处理函数向量和向量变量的链式法则了。令人意外的是，这条更一般的法则，看起来与标量的单变量链式法则同样简洁。与其直接给出公式，不如亲自重新推导，以便牢固掌握。先计算一个向量函数对标量的导数，例如 $\mathbf{y} = \mathbf{f}(x)$ ，看看能否从中抽象出一般公式。

$$\begin{bmatrix} y_1(x) \\ y_2(x) \end{bmatrix} = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} \ln(x^2) \\ \sin(3x) \end{bmatrix}$$

引入两个中间变量 g_1 和 g_2 ，每个 f_i 对应一个，使 $\mathbf{y}$ 更像 $\mathbf{y} = \mathbf{f}(\mathbf{g}(x))$ ：

$$\begin{bmatrix} g_1(x) \\ g_2(x) \end{bmatrix} = \begin{bmatrix} x^2 \\ 3x \end{bmatrix}$$

$$\begin{bmatrix} f_1(\mathbf{g}) \\ f_2(\mathbf{g}) \end{bmatrix} = \begin{bmatrix} \ln(g_1) \\ \sin(g_2) \end{bmatrix}$$

向量 $\mathbf{y}$ 对标量 x 的导数是一个列向量，其各元素可用单变量全导数链式法则计算：

$$\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial f_1(\mathbf{g})}{\partial x} \\ \frac{\partial f_2(\mathbf{g})}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial g_1} \frac{\partial g_1}{\partial x} + \frac{\partial f_1}{\partial g_2} \frac{\partial g_2}{\partial x} \\ \frac{\partial f_2}{\partial g_1} \frac{\partial g_1}{\partial x} + \frac{\partial f_2}{\partial g_2} \frac{\partial g_2}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{1}{g_1} 2x + 0 \\ 0 + \cos(g_2) 3 \end{bmatrix} = \begin{bmatrix} \frac{2x}{x^2} \\ 3\cos(3x) \end{bmatrix} = \begin{bmatrix} \frac{2}{x} \\ 3\cos(3x) \end{bmatrix}$$

现在，我们仅靠标量法则就得到了答案，只不过把导数组织成了向量。接下来尝试从这一结果抽象出向量形式：目标是将下面由标量运算组成的向量，改写为一次向量运算。

$$\begin{bmatrix} \frac{\partial f_1}{\partial g_1} \frac{\partial g_1}{\partial x} + \frac{\partial f_1}{\partial g_2} \frac{\partial g_2}{\partial x} \\ \frac{\partial f_2}{\partial g_1} \frac{\partial g_1}{\partial x} + \frac{\partial f_2}{\partial g_2} \frac{\partial g_2}{\partial x} \end{bmatrix}$$

将 $\frac{\partial f_i}{\partial g_j} \frac{\partial g_j}{\partial x}$ 各项拆开，并把 $\frac{\partial g_j}{\partial x}$ 各项单独组成一个向量，就得到矩阵与向量的乘法：

$$\begin{bmatrix} \frac{\partial f_1}{\partial g_1} & \frac{\partial f_1}{\partial g_2} \\ \frac{\partial f_2}{\partial g_1} & \frac{\partial f_2}{\partial g_2} \end{bmatrix} \begin{bmatrix} \frac{\partial g_1}{\partial x} \\ \frac{\partial g_2}{\partial x} \end{bmatrix} = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial x}$$

这意味着，一个雅可比矩阵等于另外两个雅可比矩阵的乘积，很有意思。来核对一下结果：

$$\frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial x} = \begin{bmatrix} \frac{1}{g_1} & 0 \\ 0 & \cos(g_2) \end{bmatrix} \begin{bmatrix} 2x \\ 3 \end{bmatrix} = \begin{bmatrix} \frac{1}{g_1} 2x + 0 \\ 0 + \cos(g_2) 3 \end{bmatrix} = \begin{bmatrix} \frac{2}{x} \\ 3\cos(3x) \end{bmatrix}$$

太好了！结果与标量方法相同。对于函数向量和单个参数，这条向量链式法则看来是正确的，而且确实与单变量链式法则相呼应。比较向量法则：

$$\frac{\partial}{\partial x} \mathbf{f}(\mathbf{g}(x)) = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial x}$$

与单变量链式法则：

$$\frac{d}{dx} f(g(x)) = \frac{df}{dg} \frac{dg}{dx}$$

要使公式适用于多个参数或向量 $\mathbf{x}$ ，只需将式中的 x 换成向量 $\mathbf{x}$ 。这样， $\frac{\partial \mathbf{g}}{\partial \mathbf{x}}$ 和最终得到的雅可比矩阵 $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ 就从列向量变成了矩阵。完整的向量链式法则为：

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{g}(\mathbf{x})) = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \quad (\text{Note: matrix multiply doesn't commute; order of } \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \text{ matters})$$

相比单变量链式法则，向量公式的妙处在于：保持同样简洁的符号形式，同时自动计入全导数。雅可比矩阵包含了 f_i 对 g_j ，以及 g_i 对 x_j 的所有可能组合。为完整起见，下面展开这两个雅可比矩阵：

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{g}(\mathbf{x})) = \begin{bmatrix} \frac{\partial f_1}{\partial g_1} & \frac{\partial f_1}{\partial g_2} & \cdots & \frac{\partial f_1}{\partial g_k} \\ \frac{\partial f_2}{\partial g_1} & \frac{\partial f_2}{\partial g_2} & \cdots & \frac{\partial f_2}{\partial g_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial g_1} & \frac{\partial f_m}{\partial g_2} & \cdots & \frac{\partial f_m}{\partial g_k} \end{bmatrix} \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_k}{\partial x_1} & \frac{\partial g_k}{\partial x_2} & \cdots & \frac{\partial g_k}{\partial x_n} \end{bmatrix}$$

其中 $m = |f|$ 、 $n = |x|$ 、 $k = |g|$ 。所得雅可比矩阵大小为 $m \times n$ (一个 $m \times k$ 矩阵乘以一个 $k \times n$ 矩阵)。

即使在 $\frac{\partial f}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$ 这个公式中，也常能进一步简化，因为在许多应用里，雅可比矩阵是方阵 ($m = n$)，且非对角元素为零。神经网络中的数学通常处理的是以向量为输入的函数，而不是一组任意函数组成的向量。例如，神经元的仿射函数包含 $\text{sum}(\mathbf{w} \otimes \mathbf{x})$ 项，激活函数为 $\max(0, \mathbf{x})$ ；下一节将讨论它们的导数。

前面已经看到，对向量 $\mathbf{w}$ 和 $\mathbf{x}$ 的逐元素运算会产生对角矩阵，其元素为 $\frac{\partial w_i}{\partial x_i}$ ，因为 w_i 仅依赖 x_i ，不依赖 $j \neq i$ 时的 x_j 。这里也一样：当 f_i 仅是 g_i 的函数，且 g_i 仅是 x_i 的函数时，有：

$$\frac{\partial \mathbf{f}}{\partial \mathbf{g}} = \text{diag}\left(\frac{\partial f_i}{\partial g_i}\right)$$

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \text{diag}\left(\frac{\partial g_i}{\partial x_i}\right)$$

在这种情况下，向量链式法则简化为：

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{g}(\mathbf{x})) = \text{diag}\left(\frac{\partial f_i}{\partial g_i}\right) \text{diag}\left(\frac{\partial g_i}{\partial x_i}\right) = \text{diag}\left(\frac{\partial f_i}{\partial g_i} \frac{\partial g_i}{\partial x_i}\right)$$

因此，雅可比矩阵化为对角矩阵，每个对角元素都是相应单变量链式法则的计算结果。

经过这一大段数学推导，终于得到回报：你只需要掌握向量链式法则，因为单变量公式都是它的特例。下表汇总了为得到雅可比矩阵而需要相乘的各个组成部分。

	scalar		vector		vector	
	x		u		x	
	scalar		vector		vector	
	u		u		u	
$\frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{g}(\mathbf{x})) = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$						
scalar						
f	$\frac{\partial f}{\partial u}$	$\frac{\partial u}{\partial x}$	$\frac{\partial f}{\partial u}$	$\frac{\partial u}{\partial x}$	$\frac{\partial f}{\partial u}$	$\frac{\partial u}{\partial x}$
vector						
$\mathbf{f}$	$\frac{\partial \mathbf{f}}{\partial u}$	$\frac{\partial \mathbf{u}}{\partial x}$	$\frac{\partial \mathbf{f}}{\partial u}$	$\frac{\partial \mathbf{u}}{\partial x}$	$\frac{\partial \mathbf{f}}{\partial u}$	$\frac{\partial \mathbf{u}}{\partial x}$

神经元激活值的梯度

现在，我们已经具备所需的全部工具，可以计算神经网络中单个计算单元的典型激活值对模型参数 $\mathbf{w}$ 和 b 的导数：

$$\text{activation}(\mathbf{x}) = \max(0, \mathbf{w} \cdot \mathbf{x} + b)$$

(这表示一个采用全连接权重和整流线性单元激活函数的神经元。当然，也存在卷积等其他仿射函数，以及指数线性单元等其他激活函数，它们遵循类似的逻辑。)

先暂时放下 $\max$ ，专注于计算 $\frac{\partial}{\partial \mathbf{w}}(\mathbf{w} \cdot \mathbf{x} + b)$ 和 $\frac{\partial}{\partial b}(\mathbf{w} \cdot \mathbf{x} + b)$ 。（回忆一下，神经网络通过优化权重和偏置来学习。）我们还没有讨论点积的导数 $y = \mathbf{f}(\mathbf{w}) \cdot \mathbf{g}(\mathbf{x})$ ，但可以利用链式法则，避免再记一条新规则。（注意，这里使用 y 而非 $\mathbf{y}$ ，因为结果是标量，不是向量。）

点积 $\mathbf{w} \cdot \mathbf{x}$ 就是逐元素乘积的和： $\sum_i^n (w_i x_i) = \text{sum}(\mathbf{w} \otimes \mathbf{x})$ 。（记住线性代数形式 $\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x}$ 也可能有所帮助。）我们知道如何计算 $\text{sum}(\mathbf{x})$ 和 $\mathbf{w} \otimes \mathbf{x}$ 的偏导数，但还没有讨论 $\text{sum}(\mathbf{w} \otimes \mathbf{x})$ 的偏导数。这需要链式法则，因此可以像使用单变量链式法则时那样，引入中间向量变量 $\mathbf{u}$ ：

$$\begin{aligned}\mathbf{u} &= \mathbf{w} \otimes \mathbf{x} \\ y &= \text{sum}(\mathbf{u})\end{aligned}$$

将 y 改写之后，就能认出两个偏导数已经知道的子表达式：

$$\begin{aligned}\frac{\partial \mathbf{u}}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}(\mathbf{w} \otimes \mathbf{x}) = \text{diag}(\mathbf{x}) \\ \frac{\partial y}{\partial \mathbf{u}} &= \frac{\partial}{\partial \mathbf{u}} \text{sum}(\mathbf{u}) = \vec{1}^T\end{aligned}$$

向量链式法则告诉我们，将这些偏导数相乘：

$$\frac{\partial y}{\partial \mathbf{w}} = \frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{w}} = \vec{1}^T \text{diag}(\mathbf{x}) = \mathbf{x}^T$$

为了核对结果，可以将点积完全展开为标量函数：

$$\begin{aligned}y &= \mathbf{w} \cdot \mathbf{x} = \sum_i^n (w_i x_i) \\ \frac{\partial y}{\partial w_j} &= \frac{\partial}{\partial w_j} \sum_i (w_i x_i) = \sum_i \frac{\partial}{\partial w_j} (w_i x_i) = \frac{\partial}{\partial w_j} (w_j x_j) = x_j\end{aligned}$$

于是：

$$\frac{\partial y}{\partial \mathbf{w}} = [x_1, \dots, x_n] = \mathbf{x}^T$$

很好！标量计算的结果与向量链式法则的结果一致。

现在令 $y = \mathbf{w} \cdot \mathbf{x} + b$ ，也就是 $\max$ 激活函数调用内部的完整表达式。我们要计算两个不同的偏导数，但这次不需要链式法则：

$$\begin{aligned}\frac{\partial y}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \mathbf{w} \cdot \mathbf{x} + \frac{\partial}{\partial \mathbf{w}} b = \mathbf{x}^T + \vec{0}^T = \mathbf{x}^T \\ \frac{\partial y}{\partial b} &= \frac{\partial}{\partial b} \mathbf{w} \cdot \mathbf{x} + \frac{\partial}{\partial b} b = 0 + 1 = 1\end{aligned}$$

接下来处理神经元激活值 $\max(0, \mathbf{w} \cdot \mathbf{x} + b)$ 的偏导数。对标量 z 应用 $\max(0, z)$ ，就是将所有负的 z 值视为 0。 $\max$ 函数的导数是分段函数。当 $z \leq 0$ 时，导数为 0，因为 z 是常数；当 $z > 0$ 时， $\max$ 的导数就是 z 的导数，也就是 1：

$$\frac{\partial}{\partial z} \max(0, z) = \begin{cases} 0 & z \leq 0 \\ \frac{dz}{dz} = 1 & z > 0 \end{cases}$$

补充说明如何将标量函数广播到各元素上。当 $\max$ 的一个或两个参数是向量时，例如 $\max(0, \mathbf{x})$ ，我们将单变量函数 $\max$ 广播到各个元素。这是逐元素一元运算的一个例子。明确写出就是：

$$\max(0, \mathbf{x}) = \begin{bmatrix} \max(0, x_1) \\ \max(0, x_2) \\ \vdots \\ \max(0, x_n) \end{bmatrix}$$

对于广播后的版本，其导数对应一个由 0 和 1 组成的向量，其中：

$$\frac{\partial}{\partial x_i} \max(0, x_i) = \begin{cases} 0 & x_i \leq 0 \\ \frac{dx_i}{dx_i} = 1 & x_i > 0 \end{cases}$$

$$\frac{\partial}{\partial \mathbf{x}} \max(0, \mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} \max(0, x_1) \\ \frac{\partial}{\partial x_2} \max(0, x_2) \\ \vdots \\ \frac{\partial}{\partial x_n} \max(0, x_n) \end{bmatrix}$$

要计算 $\text{activation}(\mathbf{x})$ 的导数，由于存在嵌套子表达式 $\mathbf{w} \cdot \mathbf{x} + b$ ，需要使用链式法则。按照既定流程，引入中间标量变量 z 表示仿射函数，得到：

$$z(\mathbf{w}, b, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

$$\text{activation}(z) = \max(0, z)$$

向量链式法则给出：

$$\frac{\partial \text{activation}}{\partial \mathbf{w}} = \frac{\partial \text{activation}}{\partial z} \frac{\partial z}{\partial \mathbf{w}}$$

可以将其改写为：

$$\frac{\partial \text{activation}}{\partial \mathbf{w}} = \begin{cases} 0 \frac{\partial z}{\partial \mathbf{w}} = \vec{0}^T & z \leq 0 \\ 1 \frac{\partial z}{\partial \mathbf{w}} = \frac{\partial z}{\partial \mathbf{w}} = \mathbf{x}^T & z > 0 \end{cases} \quad (\text{we computed } \frac{\partial z}{\partial \mathbf{w}} = \mathbf{x}^T \text{ previously})$$

再将 $z = \mathbf{w} \cdot \mathbf{x} + b$ 代回：

$$\frac{\partial \text{activation}}{\partial \mathbf{w}} = \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ \mathbf{x}^T & \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

这个等式符合直觉。当激活函数将仿射函数输出 z 截断为 0 时，对任意权重 w_i 的导数都为零。当 $z > 0$ 时， $\max$ 函数仿佛消失了一样，剩下的就是 z 对权重的导数。

接着计算神经元激活值对 b 的导数，得到：

$$\frac{\partial \text{activation}}{\partial b} = \begin{cases} 0 \frac{\partial z}{\partial b} = 0 & \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 \frac{\partial z}{\partial b} = 1 & \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

现在利用这些偏导数，处理整个损失函数。

神经网络损失函数的梯度

训练神经元，需要对损失函数（也称“代价函数”）关于模型参数 $\mathbf{w}$ 和 b 求导。本例采用均方误差作为损失函数。由于训练使用多个向量输入（例如多张图像）及相应的标量目标（例如每张图像对应一个分类），我们需要引入更多符号。令：

$$X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^T$$

其中 $N = |X|$ ，再令：

$$\mathbf{y} = [\text{target}(\mathbf{x}_1), \text{target}(\mathbf{x}_2), \dots, \text{target}(\mathbf{x}_N)]^T$$

其中 y_i 是标量。于是代价函数变为：

$$C(\mathbf{w}, b, X, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^N (y_i - \text{activation}(\mathbf{x}_i))^2 = \frac{1}{N} \sum_{i=1}^N (y_i - \max(0, \mathbf{w} \cdot \mathbf{x}_i + b))^2$$

按照链式法则流程，引入以下中间变量：

$$\begin{aligned} u(\mathbf{w}, b, \mathbf{x}) &= \max(0, \mathbf{w} \cdot \mathbf{x} + b) \\ v(y, u) &= y - u \\ C(v) &= \frac{1}{N} \sum_{i=1}^N v^2 \end{aligned}$$

先计算对 $\mathbf{w}$ 的梯度。

对权重的梯度

根据前面的结果，我们知道：

$$\frac{\partial}{\partial \mathbf{w}} u(\mathbf{w}, b, \mathbf{x}) = \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ \mathbf{x}^T & \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

以及

$$\frac{\partial v(y, u)}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}} (y - u) = \vec{0}^T - \frac{\partial u}{\partial \mathbf{w}} = -\frac{\partial u}{\partial \mathbf{w}} = \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ -\mathbf{x}^T & \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

于是，整体梯度为：

$$\begin{aligned}
\frac{\partial C(v)}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \frac{1}{N} \sum_{i=1}^N v^2 \\
&= \frac{1}{N} \sum_{i=1}^N \frac{\partial}{\partial \mathbf{w}} v^2 \\
&= \frac{1}{N} \sum_{i=1}^N \frac{\partial v^2}{\partial v} \frac{\partial v}{\partial \mathbf{w}} \\
&= \frac{1}{N} \sum_{i=1}^N 2v \frac{\partial v}{\partial \mathbf{w}} \\
&= \frac{1}{N} \sum_{i=1}^N \begin{cases} 2v \vec{0}^T = \vec{0}^T & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ -2v \mathbf{x}_i^T & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\
&= \frac{1}{N} \sum_{i=1}^N \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ -2(y_i - u) \mathbf{x}_i^T & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\
&= \frac{1}{N} \sum_{i=1}^N \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ -2(y_i - \max(0, \mathbf{w} \cdot \mathbf{x}_i + b)) \mathbf{x}_i^T & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\
&= \frac{1}{N} \sum_{i=1}^N \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ -2(y_i - (\mathbf{w} \cdot \mathbf{x}_i + b)) \mathbf{x}_i^T & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\
&= \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ \frac{-2}{N} \sum_{i=1}^N (y_i - (\mathbf{w} \cdot \mathbf{x}_i + b)) \mathbf{x}_i^T & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\
&= \begin{cases} \vec{0}^T & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ \frac{2}{N} \sum_{i=1}^N (\mathbf{w} \cdot \mathbf{x}_i + b - y_i) \mathbf{x}_i^T & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases}
\end{aligned}$$

为了理解这个等式，可以代入误差项 $e_i = \mathbf{w} \cdot \mathbf{x}_i + b - y_i$ ，得到：

$$\frac{\partial C}{\partial \mathbf{w}} = \frac{2}{N} \sum_{i=1}^N e_i \mathbf{x}_i^T \quad (\text{for the nonzero activation cases})$$

注意，这个计算是在 X 中的所有 $\mathbf{x}_i$ 上取加权平均。权重就是误差项，即对每个输入 $\mathbf{x}_i$ ，目标输出与神经元实际输出之间的差。所得梯度总体上指向代价或损失增大的方向，因为较大的 e_i 会突出对应的 $\mathbf{x}_i$ 的贡献。假设只有一个输入向量 $N = |X| = 1$ ，梯度就是 $2e_1 \mathbf{x}_1^T$ 。误差为 0 时，梯度为零，损失也就达到了最小值。如果 e_1 是一个较小的正差值，梯度就是沿 $\mathbf{x}_1$ 方向迈出一小步；如果 e_1 很大，梯度就是沿该方向迈出一大步。如果 e_1 为负，梯度方向就反过来，意味着代价增大的方向是负方向。

当然，我们希望减小而不是增大损失，因此**梯度下降**的递推公式沿负梯度方向更新当前位置（标量学习率为 η ）：

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{\partial C}{\partial \mathbf{w}}$$

由于梯度指向代价增大的方向，我们要沿相反方向更新 $\mathbf{w}$ 。

对偏置的导数

为了优化偏置 b ，还需要对 b 的偏导数。再次列出中间变量：

$$\begin{aligned} u(\mathbf{w}, b, \mathbf{x}) &= \max(0, \mathbf{w} \cdot \mathbf{x} + b) \\ v(y, u) &= y - u \\ C(v) &= \frac{1}{N} \sum_{i=1}^N v^2 \end{aligned}$$

前面已经计算了方程 $u(\mathbf{w}, b, \mathbf{x})$ 对偏置的偏导数：

$$\frac{\partial u}{\partial b} = \begin{cases} 0 & \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

对于 v ，偏导数为：

$$\frac{\partial v(y, u)}{\partial b} = \frac{\partial}{\partial b}(y - u) = 0 - \frac{\partial u}{\partial b} = -\frac{\partial u}{\partial b} = \begin{cases} 0 & \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ -1 & \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

而代价函数本身的偏导数为：

$$\begin{aligned} \frac{\partial C(v)}{\partial b} &= \frac{\partial}{\partial b} \frac{1}{N} \sum_{i=1}^N v^2 \\ &= \frac{1}{N} \sum_{i=1}^N \frac{\partial}{\partial b} v^2 \\ &= \frac{1}{N} \sum_{i=1}^N \frac{\partial v^2}{\partial v} \frac{\partial v}{\partial b} \\ &= \frac{1}{N} \sum_{i=1}^N 2v \frac{\partial v}{\partial b} \\ &= \frac{1}{N} \sum_{i=1}^N \begin{cases} 0 & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ -2v & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\ &= \frac{1}{N} \sum_{i=1}^N \begin{cases} 0 & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ -2(y_i - \max(0, \mathbf{w} \cdot \mathbf{x}_i + b)) & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\ &= \frac{1}{N} \sum_{i=1}^N \begin{cases} 0 & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ 2(\mathbf{w} \cdot \mathbf{x}_i + b - y_i) & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \\ &= \frac{2}{N} \sum_{i=1}^N \begin{cases} 0 & \mathbf{w} \cdot \mathbf{x}_i + b \leq 0 \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i & \mathbf{w} \cdot \mathbf{x}_i + b > 0 \end{cases} \end{aligned}$$

和之前一样，可以代入误差项：

$$\frac{\partial C}{\partial b} = \frac{2}{N} \sum_{i=1}^N e_i \quad (\text{for the nonzero activation case})$$

因此，偏导数就是根据激活情况取误差值或零，再对它们求平均。更新神经元偏置时，沿代价增大的相反方向作小幅调整：

$$b_{t+1} = b_t - \eta \frac{\partial C}{\partial b}$$

实际中，将 $\mathbf{w}$ 与 b 合并为一个向量参数更方便，这样不必分别处理两个偏导数： $\hat{\mathbf{w}} = [\mathbf{w}^T, b]^T$ 。这也需要调整输入向量 $\mathbf{x}$ ，但能简化激活函数。在 $\mathbf{x}$ 的末尾补上一个 1，得到 $\hat{\mathbf{x}} = [\mathbf{x}^T, 1]$ 后， $\mathbf{w} \cdot \mathbf{x} + b$ 就变为 $\hat{\mathbf{w}} \cdot \hat{\mathbf{x}}$ 。

至此，神经网络损失函数的优化推导就完成了：我们已经得到执行梯度下降所需的两个偏导数。

总结

希望你一路读到了这里。你已经在理解矩阵微积分的道路上迈出了重要一步！下一节提供速查资料，汇总本文的全部法则。也请看看后面附带评述的资源链接。

接下来，可以学习对矩阵而不只是对向量求偏导。例如，阅读 [Matrix calculus](#) 中关于矩阵微分的部分。

致谢。感谢 [Yannet Interian](#)（旧金山大学数据科学硕士项目教师）和 [David Uminsky](#)（数据科学硕士项目教师兼主任）帮助完善本文的符号记法。

矩阵微积分速查

梯度与雅可比矩阵

二元函数的梯度是一个长度为 2 的行向量：

$$\nabla f(x,y) = [\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}]$$

以向量为输入的向量值函数，其雅可比矩阵是一个包含全部可能标量偏导数的 $m \times n$ 矩阵 ($m = |f|, n = |x|$)：

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \nabla f_1(x) \\ \nabla f_2(x) \\ \dots \\ \nabla f_m(x) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f_1(x) \\ \frac{\partial}{\partial x} f_2(x) \\ \dots \\ \frac{\partial}{\partial x} f_m(x) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(x) \frac{\partial}{\partial x_2} f_1(x) \dots \frac{\partial}{\partial x_n} f_1(x) \\ \frac{\partial}{\partial x_1} f_2(x) \frac{\partial}{\partial x_2} f_2(x) \dots \frac{\partial}{\partial x_n} f_2(x) \\ \dots \\ \frac{\partial}{\partial x_1} f_m(x) \frac{\partial}{\partial x_2} f_m(x) \dots \frac{\partial}{\partial x_n} f_m(x) \end{bmatrix}$$

恒等函数 $f(x) = x$ 的雅可比矩阵为 I 。

向量的逐元素运算

使用 $\odot$ 这样的运算符（例如 $+$ ），定义向量 w 和 x 上的一般逐元素运算：

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f_1(w) \odot g_1(x) \\ f_n(w) \odot g_2(x) \\ \vdots \\ f_n(w) \odot g_n(x) \end{bmatrix}$$

对 w 的雅可比矩阵为（对 x 的情形类似）：

$$J_w = \frac{\partial y}{\partial w} = \begin{bmatrix} \frac{\partial}{\partial w_1}(f_1(w) \odot g_1(x)) & \frac{\partial}{\partial w_2}(f_1(w) \odot g_1(x)) & \dots & \frac{\partial}{\partial w_n}(f_1(w) \odot g_1(x)) \\ \frac{\partial}{\partial w_1}(f_2(w) \odot g_2(x)) & \frac{\partial}{\partial w_2}(f_2(w) \odot g_2(x)) & \dots & \frac{\partial}{\partial w_n}(f_2(w) \odot g_2(x)) \\ \dots & \dots & \dots & \dots \\ \frac{\partial}{\partial w_1}(f_n(w) \odot g_n(x)) & \frac{\partial}{\partial w_2}(f_n(w) \odot g_n(x)) & \dots & \frac{\partial}{\partial w_n}(f_n(w) \odot g_n(x)) \end{bmatrix}$$

如果满足逐元素对角条件，即 $f_i(w)$ 和 $g_i(x)$ 分别最多只访问 w_i 和 x_i ，则雅可比矩阵化为对角矩阵：

$$\frac{\partial y}{\partial w} = diag \left(\frac{\partial}{\partial w_1}(f_1(w_1) \odot g_1(x_1)), \frac{\partial}{\partial w_2}(f_2(w_2) \odot g_2(x_2)), \dots, \frac{\partial}{\partial w_n}(f_n(w_n) \odot g_n(x_n)) \right)$$

以下是一些逐元素运算的例子：

	Partial with respect to w	Partial with respect to x
$+$	$\frac{\partial (w+x)}{\partial w} = I$	$\frac{\partial (w+x)}{\partial x} = I$
$-$	$\frac{\partial (w-x)}{\partial w} = I$	$\frac{\partial (w-x)}{\partial x} = -I$
$\otimes$	$\frac{\partial (w \otimes x)}{\partial w} = diag(x)$	$\frac{\partial (w \otimes x)}{\partial x} = diag(w)$
$\oslash$	$\frac{\partial (w \oslash x)}{\partial w} = diag(\dots \frac{1}{x_i} \dots)$	$\frac{\partial (w \oslash x)}{\partial x} = diag(\dots \frac{-w_i}{x_i^2} \dots)$

标量扩展

将标量 z 加到向量 $\mathbf{x}$ 上得到 $y = \mathbf{x} + z$ ，实际上就是 $y = f(\mathbf{x}) + g(z)$ ，其中 $f(\mathbf{x}) = \mathbf{x}$ 且 $g(z) = \vec{1}z$ 。

$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{x} + z) = \text{diag}(\vec{1}) = I$$

$$\frac{\partial}{\partial z}(\mathbf{x} + z) = \vec{1}$$

标量乘法得到：

$$\frac{\partial}{\partial \mathbf{x}}(\mathbf{x}z) = Iz$$

$$\frac{\partial}{\partial z}(\mathbf{x}z) = \mathbf{x}$$

向量归约

向量求和对其中一个向量的偏导数为：

$$\nabla_{\mathbf{x}} y = \frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n} \right] = \left[\sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_1}, \sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_2}, \dots, \sum_i \frac{\partial f_i(\mathbf{x})}{\partial x_n} \right]$$

对于 $y = \text{sum}(\mathbf{x})$ ：

$$\nabla_{\mathbf{x}} y = \vec{1}^T$$

对于 $y = \text{sum}(\mathbf{x}z)$ 和 $n = |\mathbf{x}|$ ，得到：

$$\nabla_{\mathbf{x}} y = [z, z, \dots, z]$$

$$\nabla_z y = \text{sum}(\mathbf{x})$$

向量点积为 $y = f(\mathbf{w}) \cdot g(\mathbf{x}) = \sum_i^n (w_i x_i) = \text{sum}(\mathbf{w} \otimes \mathbf{x})$ 。代入 $\mathbf{u} = \mathbf{w} \otimes \mathbf{x}$ 并使用向量链式法则，得到：

$$\begin{aligned} \frac{d\mathbf{u}}{d\mathbf{x}} &= \frac{d}{d\mathbf{x}}(\mathbf{w} \otimes \mathbf{x}) = \text{diag}(\mathbf{w}) \\ \frac{dy}{d\mathbf{x}} &= \frac{d}{d\mathbf{u}} \text{sum}(\mathbf{u}) = \vec{1}^T \\ \frac{dy}{d\mathbf{x}} &= \frac{dy}{d\mathbf{u}} \times \frac{d\mathbf{u}}{d\mathbf{x}} = \vec{1}^T \times \text{diag}(\mathbf{w}) = \mathbf{w}^T \end{aligned}$$

类似地， $\frac{dy}{d\mathbf{w}} = \mathbf{x}^T$ 。

链式法则

向量链式法则是—般形式，其他法则都是它的退化情形。当 f 是单变量 x 的函数，且所有中间变量 u 都是单变量函数时，适用单变量链式法则。当部分或全部中间变量是多变量函数时，适用单变量全导数链式法则。其他情况下，则使用向量链式法则。

单变量法则	单变量全导数法则	向量法则
$\frac{df}{dx} = \frac{df}{du} \frac{du}{dx}$	$\frac{\partial f(u_1, \dots, u_n)}{\partial x} = \frac{\partial f}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial x}$	$\frac{\partial}{\partial \mathbf{x}} f(\mathbf{g}(\mathbf{x})) = \frac{\partial f}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$

符号说明

像 $\mathbf{x}$ 这样的粗体小写字母表示向量，像 x 这样的斜体字母表示标量。 x_i 是向量 $\mathbf{x}$ 的第 i^{th} 个元素，使用斜体是因为单个向量元素是标量。 $|\mathbf{x}|$ 表示“向量 $\mathbf{x}$ 的长度”。

$\mathbf{x}^T$ 中的上标 T 表示该向量的转置。

$\sum_{i=a}^b x_i$ 相当于一个 for 循环：让 i 从 a 遍历到 b ，将所有 x_i 相加。

记号 $f(x)$ 表示名为 f 、参数为 x 的函数。

I 表示维度适当的方形“单位矩阵”：主对角线全为 1，其他位置全为零。

$\text{diag}(\mathbf{x})$ 构造一个对角元素取自向量 $\mathbf{x}$ 的矩阵。

点积 $\mathbf{w} \cdot \mathbf{x}$ 是逐元素乘积的和： $\sum_i^n (w_i x_i) = \text{sum}(\mathbf{w} \otimes \mathbf{x})$ 。也可以将其看作 $\mathbf{w}^T \mathbf{x}$ 。

求导 $\frac{d}{dx}$ 是将单参数函数映射到另一个函数的算子。也就是说， $\frac{d}{dx} f(x)$ 把 $f(x)$ 映射为它对 x 的导数，即 $\frac{df(x)}{dx}$ 。另外，如果 $y = f(x)$ ，那么 $\frac{dy}{dx} = \frac{df(x)}{dx} = \frac{d}{dx} f(x)$ 。

函数对 x 的偏导数 $\frac{\partial}{\partial x} f(x)$ ，就是将其他所有变量保持不变后，进行普通的标量求导。

f 对向量 $\mathbf{x}$ 的梯度 $\nabla f(\mathbf{x})$ ，将一个特定标量函数的全部偏导数组织起来。

雅可比矩阵通过逐行堆叠，将多个函数的梯度组织成一个矩阵：

$$J = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \end{bmatrix}$$

下面的记号表示：当 condition_1 时， y 取值 a ；当 condition_2 时，取值 b 。

$$y = \begin{cases} a & \text{condition}_1 \\ b & \text{condition}_2 \end{cases}$$

参考资料

[Wolfram Alpha](#) 可以进行符号矩阵代数运算；另外，还有一个很不错的专用[矩阵微积分求导工具](#)。

在网上查找资料时，请搜索“matrix calculus”，而不是“vector calculus”。下面是对 [Google 搜索](#) 中靠前结果的一些评述：

- https://en.wikipedia.org/wiki/Matrix_calculus

Wikipedia 的条目其实相当不错，对不同布局约定也有清楚的说明。回忆一下，我们采用分子布局：雅可比矩阵中的变量横向排列，函数纵向排列。Wikipedia 对[全导数](#)的解释也很好，但要注意，其记号与本文略有不同。我们始终使用 ∂x ，而不是 dx 。

- <http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/calculus.html>

这个网页有一节讨论矩阵微分，给出了一些有用的恒等式，作者采用分子布局。读完本文后，如果想进一步了解矩阵求导与向量求导的区别，可以从这里开始。

- <https://www.colorado.edu/engineering/CAS/courses.d/IFEM.d/IFEM.AppC.d/IFEM.AppC.pdf>

这是《有限元方法导论》课程讲义的一部分，作者应该是 [Carlos A. Felippa](#)。他采用分母布局，因此其雅可比矩阵是本文记法的转置。

- <http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/calculus.html>

这个网页列出了大量不同向量和矩阵的实用导数，是一份很好的速查表。基本没有解释，只是一组法则。

- <https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf>

另一份速查资料，主要讨论一般矩阵运算，比上一项有更多说明。

- <https://www.comp.nus.edu.sg/~cs5240/lecture/matrix-differentiation.pdf>

一组有用的幻灯片。

如果想进一步了解神经网络，以及优化和反向传播背后的数学，我们强烈推荐 [Michael Nielsen 的书](#)。

如果你特别关注卷积神经网络，可以阅读《深度学习卷积运算指南》（[A guide to convolution arithmetic for deep learning](#)）。

本文引用了[全导数](#)法则。这是一个重要概念，其含义只是：对 x 求导时，必须考虑所有依赖于 x 的变量对 x 的导数。