

King of Dumbs

笨蛋之王

级别: ♠ 技术

电脑怎样把照片变成数字？

一张照片就是几千万个数，人工智能对它做的几乎全是乘法和加法。分三个层次讲向量、点积和矩阵，配两个互动小工具。

中文 发布于 2026年10月17日



扫码打开可互动的课程

<https://kingofdumbs.com/zh/lessons/vectors-and-matrices>

你将学到

- 照片就是一列数：向量和维数
- 点积：乘一乘再加起来，以及两支箭头有多同向
- 矩阵会挪动平面：它的列、面积和剩下的维数
- 矩阵乘法：把变换串起来，为什么顺序要紧，以及神经网络在做什么

本课会出现的符号

E_a 下标：右下角的标签 (♦♦) · Σ 求和：全部加起来 (♠) · x_i 下标 i ：第 i 个元素 (♠) · $\approx$ 约等于 (♦) · $\leq \geq$ 小于等于 / 大于等于 (♠) · $|x|$ 绝对值：去掉正负号 (♠) · $\cdot$ 点：乘（或点积） (♦♦) · σ sigma：含义因课而异 (♠) · θ theta：通常是角度 (♠) · λ lambda：一个缩放用的数 (♠) · $\|x\|$ 范数：箭头的长度 (♠) · x^T 转置：把列放倒 (♠) · $\in$ 属于 (♠) · $\mathbb{R} \mathbb{Z}$ 数集 (♠)

你给猫拍了张照片，在手机相册里搜”猫”，很多时候就能把它找出来。可电脑并看不见猫：它唯一会做的，就是飞快地加数和乘数。

这一课的傻问题是：**电脑怎样把照片变成数字？** 顺便再问一句：变成数字以后，它拿这些数做什么？答案就是三个听起来像考试题的概念：向量、点积和矩阵；再加上一道算术：乘一乘，再加起来。

选一张牌：♣ 从零开始，♦ 不怕简单的公式，♠ 想看符号。随时可以换。

照片就是一列数

♦→♠从进阶到技术

♦里的说法	♠里的写法
向量	$x \in \mathbb{R}^n$, 由 n 个实数组成
分量	$x = (x_1, \dots, x_n)$
长度	欧几里得范数 $\ x\ = \sqrt{(\sum_i x_i^2)}$
维数	$\dim \mathbb{R}^n = n$: 一组基 $e_1, \dots, e_n$ 中向量的个数
通道	C : RGB 中 $C = 3$, 取值为 $\{0, \dots, 255\}$ 的 8 位整数
高 $\times$ 宽 $\times 3$	张量 $H \times W \times C$ (HWC); torchvision 用 $C \times H \times W$
把它全部排成一行	展平 (flatten): $x \in \mathbb{R}^n$, $n = H \cdot W \cdot C$

$$x \in \mathbb{R}^n, \quad x = (x_1, \dots, x_n)$$

$$\|x\| = \sqrt{(\sum_i x_i^2)}$$

$$\text{图像} \in \{0, \dots, 255\}^{(H \times W \times C)}, \quad C = 3$$

$$\text{ToTensor}: H \times W \times C \rightarrow C \times H \times W, \quad /255$$

$$n = H \cdot W \cdot C = 4000 \cdot 3000 \cdot 3 = 3.6 \times 10^7$$

$$\dim \mathbb{R}^n = n \quad (\text{基 } e_1, \dots, e_n)$$

8 位 RGB 图像是一个取值于 $\{0, \dots, 255\}^{(H \times W \times C)}$ 的整数张量, $C = 3$: Pillow 13.0 把这种模式描述为 «RGB (3x8-bit pixels, true color)», 并说明 «an 8-bit pixel has a range of 0-255». torchvision 0.29 用 ToTensor 把它从 $[0, 255]$ 区间的 $(H \times W \times C)$ 转成 $[0.0, 1.0]$ 区间、形状为 $(C \times H \times W)$ 的浮点张量。展平后, 它是 $\mathbb{R}^n$ 中的一个向量, $n = H \cdot W \cdot C$; 对 4000×3000 的照片, $n = 3.6 \times 10^7$ 。

一个空间的维数, 是它任意一组基中向量的个数 (Austin, *Understanding Linear Algebra*, 定义 3.5.6)。标准基 $e_1, \dots, e_n$ 有 n 个向量, 所以 $\dim \mathbb{R}^n = n$, 也就是分量的个数。范数 $\|x\| = \sqrt{(\sum_i x_i^2)}$ 就是 n 维的勾股定理。

点积：乘一乘，再加起来

♦ → ♠ 从进阶到技术

♦ 里的说法	♠ 里的写法
点积	$a \cdot b = \sum_i a_i b_i$ （也写作 $\langle a, b \rangle$ 或 $a^T b$ ）
3600 万次乘法	代价 $O(n)$ ： n 次乘法、 $n - 1$ 次加法
a 的长度 $\times b$ 的长度 $\times$ 夹角的余弦	$a \cdot b = \ a\ \ b\ \cos \theta$, $\theta \in [0, \pi]$
同向时余弦为 1，成直角时为 0，相反时为 -1	$-1 \leq \cos \theta \leq 1$ ，即 $ a \cdot b \leq \ a\ \ b\ $ （柯西-施瓦茨不等式）
垂直	正交： $a \cdot b = 0$ ($a \perp b$)
余弦相似度	$\cos \theta = a \cdot b / (\ a\ \ b\)$
箭头变长了，但没有转动	对任意 $\lambda > 0$, $\cos(x, \lambda x) = 1$

$$a \cdot b = \sum_i a_i b_i = \|a\| \|b\| \cos \theta$$

$$\cos \theta = a \cdot b / (\|a\| \|b\|) \in [-1, 1]$$

$$a \perp b \Leftrightarrow a \cdot b = 0$$

$$|a \cdot b| \leq \|a\| \|b\| \quad (\text{Cauchy-Schwarz})$$

$$\cos(x, 1.2 \cdot x) = 1: \text{更亮, 方向不变}$$

$$\text{代价: } n \text{ 次乘法、} n - 1 \text{ 次加法: } O(n)$$

在 $\mathbb{R}^n$ 中, $a \cdot b = \sum_i a_i b_i$ 。OpenStax (*Calculus Volume 3*, 2.3) 用余弦定理证明了 $a \cdot b = \|a\| \|b\| \cos \theta$ (定理 2.4), 以及两个非零向量正交当且仅当 $a \cdot b = 0$ (定理 2.5)。超过三维时反过来做: 用 $\cos \theta = a \cdot b / (\|a\| \|b\|)$ 来定义夹角, 而柯西-施瓦茨不等式 $|a \cdot b| \leq \|a\| \|b\|$ 保证这个比值落在 $[-1, 1]$ 内。

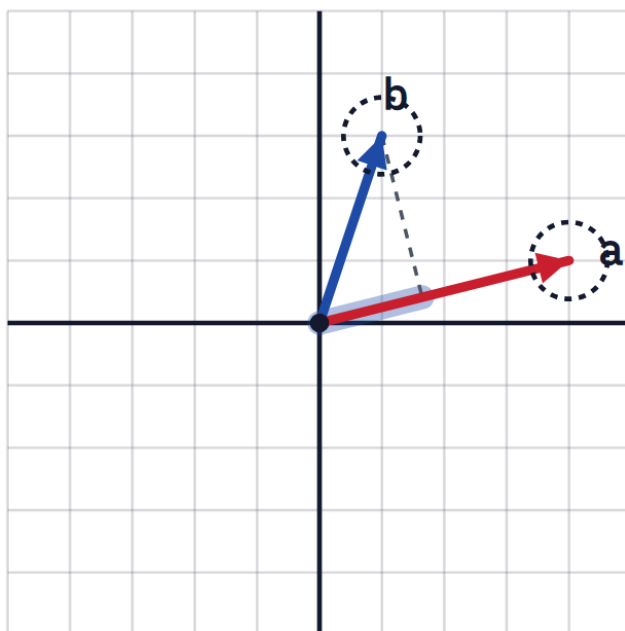
PyTorch 2.14 用 `torch.nn.functional.cosine_similarity` 计算余弦相似度, 分母取 $\max(\|x_1\| \|x_2\|, \epsilon)$, 以免除以零。因为 $\lambda > 0$ 时 $\cos(x, \lambda x) = 1$, 如果另一张照片只是把数值整体放大或缩小, 变亮或变暗, 余弦相似度就分辨不出来。代价是线性的: n 次乘法、 $n - 1$ 次加法。

笨蛋之王说:

乘一乘, 加一加, 就这么多。靠这个我能比价格、比箭头、比照片, 还剩时间睡个午觉。

动手试试：两支箭头

拖动箭头的尖端，或者用滑块、键盘方向键来移动。找一个让点积等于 0 的位置：这时两支箭头互成直角。蓝色宽带是 b 在 a 上的影子：两支箭头越同向，影子越长。



a , 横向

a , 纵向

b , 横向

b , 纵向

点积 $a \cdot b$ $4 \cdot 1 + 1 \cdot 3 = 7$

长度 $\|a\| = 4.12 \cdot \|b\| = 3.16$

夹角 57.5°

余弦（方向有多像） 0.54

$a = (4, 1)$, $b = (1, 3)$, $a \cdot b = 7$ 。指向同一侧：点积为正。



互动小工具。在网站上试一试：<https://kingofdumbs.com/zh/lessons/vectors-and-matrices>

矩阵会挪动平面

♦ → ♠ 从进阶到技术

♦ 里的说法	♠ 里的写法
矩阵	$A \in \mathbb{R}^{(m \times n)}$, 元素 a_{ij} (第 i 行第 j 列)
每一行	$(Ax)_i = \sum_j a_{ij}x_j = (A \text{ 的第 } i \text{ 行}) \cdot x$
第一列是 (1, 0) 去的地方, 第二列是 (0, 1) 去的地方	$A = [Ae_1 \ Ae_2]$: 第 j 列是 $T(e_j)$
任何一个点都是这两者的组合	线性: $A(c_1e_1 + c_2e_2) = c_1Ae_1 + c_2Ae_2$
推斜	剪切 (错切), 例如 $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$
$ad - bc$	$\det A$, 列向量所张平行四边形的有向面积
压扁	$\det A = 0$: A 不可逆; 秩下降, 像是一条直线
像照镜子一样	$\det A < 0$: 定向反转
旋转	旋转矩阵 $R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$
它来自原图的哪个点	用 A^{-1} : 遍历输出, 到输入里去找
(再加上一个平移)	仿射变换 $x \mapsto Ax + t$

$$(Ax)_i = \sum_j a_{ij} x_j$$

$$A = [Ae_1 \ Ae_2], \quad A(cu + v) = cAu + Av$$

$$R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

$\det A = ad - bc$: 有向面积

$$\det A = 0 \Leftrightarrow \text{不可逆} \Leftrightarrow \text{秩} < 2$$

$$\begin{pmatrix} 1 & -1 \\ 2 & -2 \end{pmatrix}: \det 0, \text{秩 } 1, \text{一条直线}$$

一个 $m \times n$ 矩阵 A 定义了从 $\mathbb{R}^n$ 到 $\mathbb{R}^m$ 的映射 $T(x) = Ax$, 其中 $(Ax)_i = \sum_j a_{ij}x_j$ 。它的列是基向量的像: $A = [T(e_1) \cdots T(e_n)]$ (Austin, 命题 2.5.8)。 T 是线性的, 所以 $T(\sum_j c_j e_j) = \sum_j c_j T(e_j)$, 知道了列, 就知道了一切。

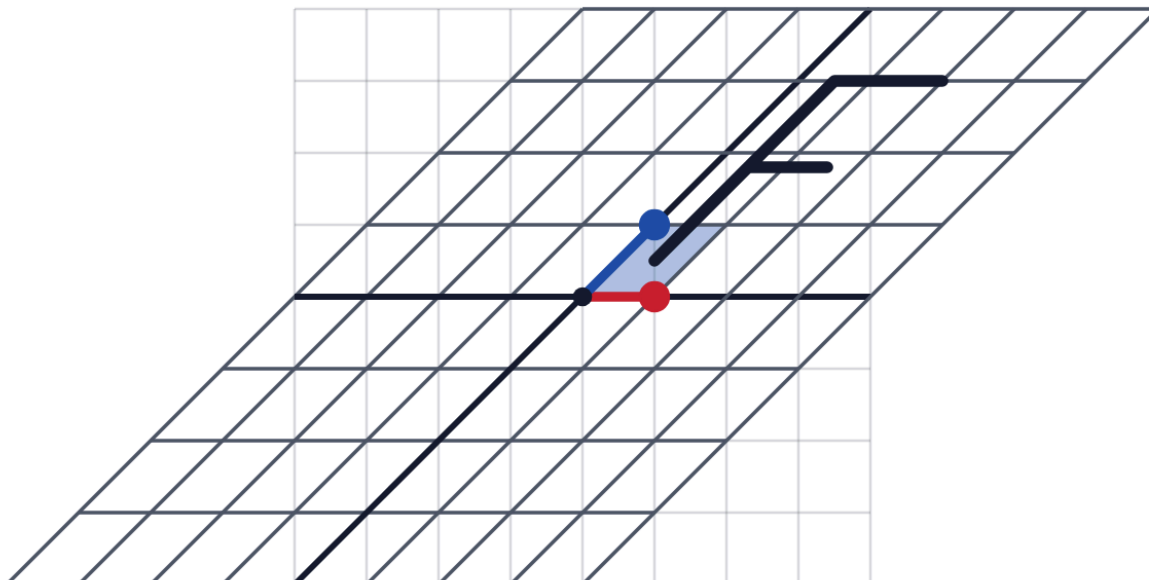
在 2×2 的情形, 转角为 θ 的旋转是 $\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$, $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ 是一个剪切 (Austin, 2.6)。
 $\det A = ad - bc$ 是列向量所张平行四边形的有向面积 (3.4), A 可逆当且仅当 $\det A \neq 0$ (命题 3.4.12)。
 $\det A = 0$ 时秩下降: 像空间 $\text{Col}(A)$ 的维数等于 $\text{rank } A$ (命题 3.5.12), 对 $\begin{pmatrix} 1 & -1 \\ 2 & -2 \end{pmatrix}$ 来说是一条直线。

Pillow 13.0 (`ImageTransform.AffineTransform`) 接收 «the first two rows from the inverse of an affine transform matrix»: 对每个输出像素 (x, y) , 到输入的 $(ax + by + c, dx + ey + f)$ 处取值。它用逆矩阵, 是因为它遍历的是输出照片; 平移项 (c, f) 让这个变换成为仿射变换, 而不是线性变换。



动手试试：让网格变形

调一调矩阵的四个数, 或者试试下面的按钮。红色箭头是第一列, 即 $(1, 0)$ 去的地方; 蓝色箭头是第二列。找一个能把网格压成一条直线的矩阵: 面积会变成 0。再看看镜像时字母 F 会怎样。



试一个

什么也不做

拉伸

转 90°

推斜

镜像

压扁

a (左上)



1

b (右上)



1

c (左下)



0

d (右下)



1

(1, 0) 变成 (1, 0)

(0, 1) 变成 (1, 1)

面积乘以 (ad - bc) **1**

剩下的维数 2: 平面还是平面

矩阵 $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$: 面积乘以 1, 剩下 2 维。



互动小工具。在网站上试一试: <https://kingofdumbs.com/zh/lessons/vectors-and-matrices>

矩阵乘法：顺序很重要

♦→♠ 从进阶到技术

♦里的说法	♠里的写法
两个矩阵相乘	$(AB)_{ij} = \sum_k a_{ik}b_{kj}$
第一个矩阵的某一行和第二个矩阵的某一列	(A 的第 i 行) · (B 的第 j 列)
$m \times k$ 矩阵乘 $k \times n$ 矩阵，得到 $m \times n$ 矩阵	$(m \times k)(k \times n) \rightarrow m \times n$ ：内侧维数相同
AB	复合： $A(Bx) = (AB)x$ ；若 $T \mapsto A$, $S \mapsto B$ ，则 $S \circ T \mapsto BA$
AB 一般不等于 BA	不满足交换律；满足结合律： $(AB)C = A(BC)$
层	线性层 (<code>nn.Linear</code>)： $y = xA^T + b$
权重	$A \in \mathbb{R}^{(\text{out} \times \text{in})}$ ，通过学习得到；b 为偏置
1000 万次乘法	$\text{out} \cdot \text{in} = 10^3 \cdot 10^4 = 10^7$ 次乘加 (FMA)，每张照片
一个不是乘法的步骤	非线性激活函数 σ ；没有它， $A_2(A_1x) = (A_2A_1)x$ 只相当于一层
一次放进很多张照片	批量 (batch) N：一次 $(M \times K)(K \times N)$ 的 GEMM， $2 \cdot M \cdot N \cdot K$ FLOP

$$(AB)_{ij} = \sum_k a_{ik} b_{kj}$$

$$(m \times k)(k \times n) \rightarrow m \times n; \quad AB \neq BA$$

$$x \mapsto A(Bx) = (AB)x$$

$$\text{代价: } m \cdot n \cdot k \text{ FMA} = 2 \cdot m \cdot n \cdot k \text{ FLOP}$$

$$\text{nn.Linear: } y = xA^T + b, \quad A: (\text{out}, \text{in})$$

$$2 \cdot 1000 \cdot 64 \cdot 10000 = 1.28 \times 10^9 \text{ FLOP}$$

$(AB)_{ij} = \sum_k a_{ik}b_{kj}$ ，要求内侧维数相同 (OpenStax, *College Algebra 2e*, 7.5)。矩阵乘法满足结合律和分配律，但不满足交换律。若 $T(x) = Ax$, $S(x) = Bx$ ，则 $S \circ T$ 的矩阵是 BA (Austin, 命题 2.5.11)：先做的那个写在右边。

$m \cdot n$ 个元素中的每一个，都是 k 项的点积：共 $m \cdot n \cdot k$ 次乘加（FMA），即 $2 \cdot m \cdot n \cdot k$ FLOP。NVIDIA 的 GEMM 指南也这样算，每个元素都是 «a dot-product of K-element vectors»。对 1000×10000 的层、一批 64 张照片： $2 \cdot 1000 \cdot 64 \cdot 10000 = 1.28 \times 10^9$ FLOP（自行计算）。

PyTorch 2.14 把线性层写成 $y = xA^T + b$ ，每个样本占一行，权重 A 的形状是 (out_features, in_features)。因为有偏置 b ，它是仿射的，不是线性的。NVIDIA 把权重放在 A 里、激活值放在 B 里，每个样本占一列：只是转置了一下，算是同一件事。层与层之间如果没有非线性的 σ ，由于结合律，连着的几层就会合成一层。

笨蛋之王说：

我看不见猫，只看见 3600 万个数。难的是知道该拿哪些矩阵去乘它们。



最后挑战

1. 一张 100×100 像素的彩色照片。电脑打开它时要存多少个数？

- a. 100
- b. 10000，每个像素一个
- c. 30000：每个像素三个，红、绿、蓝

2. 两支箭头互成直角。它们的点积是多少？

- a. 1
- b. 0
- c. 要看它们有多长

3. 把照片转 90° ，再横向拉伸。这和先拉伸、再旋转一样吗？

- a. 一样：乘法怎么换顺序都一样
- b. 不一样：矩阵相乘，顺序很重要
- c. 只有照片是正方形时才一样



三句话总结

1. 在电脑眼里，照片是一列数（每个像素三个，取值 0 到 255）：一个向量，它的维数就是数的个数。
2. 点积把两列数逐项相乘再相加：既能算出总价，就像水果摊的账；也能说明两支箭头有多同向（垂直时为 0）。
3. 矩阵是一张能挪动平面上每个点的数表，矩阵乘法把这些变换串起来，而且顺序要紧：这正是神经网络对每张照片做上百万次的运算。



参考资料

- 向量、长度和点积：OpenStax, *Calculus Volume 3*, [2.1](#) 和 [2.3](#)（水果摊的例子，定理 2.4 和 2.5）。矩阵及其乘法：OpenStax, *College Algebra 2e*, [7.5](#)。
- 变换、行列式和维数：D. Austin, *Understanding Linear Algebra*, [2.2](#)、[2.5](#)、[2.6](#)、[3.4](#) 和 [3.5](#)。
- 像素和图像变换：Pillow 13.0, [Concepts](#) 和 [ImageTransform](#)。从照片到张量：torchvision 0.29, [ToTensor](#)。
- 神经网络：PyTorch 2.14, [nn.Linear](#) 和 [cosine_similarity](#)。NVIDIA, [Matrix Multiplication Background](#) 和 [Linear/Fully-Connected Layers](#)。

这一课是怎么做的：在 Claude 的帮助下，根据上面的资料调研和撰写，资料查阅于 2026 年 10 月 17 日。没有单独链接的数字，是根据定义算出来的。例子（4000 × 3000 的照片、两家店、有 1000 个输出的层）都是编的。插图都是原创的。

答案

1. **c)** 30000: 每个像素三个, 红、绿、蓝. 对。 $100 \times 100 \times 3$: 一个 30000 维的向量。
2. **b)** 0. 答对了。互相垂直, 点积就是 0, 不管多长。
3. **b)** 不一样: 矩阵相乘, 顺序很重要. 没错。就像袜子和鞋。

King of Dumbs 的课程, kingofdumbs.com。这是离线版; 可以动手玩的原版在:

<https://kingofdumbs.com/zh/lessons/vectors-and-matrices>

更新于 2026年10月17日.