Fundamental Vector and Matrix Operations in Linear Algebra

April 6, 2025

Introduction

Linear algebra provides a concise framework for manipulating vectors and matrices. Below are the most important operations, with examples showing how to write them in LaTeX and how they can be reordered (where permissible).

1 Vectors

Let $\mathbf{u}$ and $\mathbf{v}$ be vectors in $\mathbb{R}^n$, and let α be a scalar (real number, in most common contexts). We write

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}.$$

1.1 Addition and Scalar Multiplication

$$\mathbf{u} + \mathbf{v} = \begin{pmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{pmatrix}, \quad \alpha\mathbf{u} = \begin{pmatrix} \alpha u_1 \\ \alpha u_2 \\ \vdots \\ \alpha u_n \end{pmatrix}.$$

- *Commutativity:* $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$.
- *Associativity:* $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$.

1.2 Dot (Inner) Product

$$\mathbf{u} \cdot \mathbf{v} \;=\; \mathbf{u}^\top \mathbf{v} \;=\; \sum_{i=1}^{n} u_i \, v_i.$$

- In many contexts, you will see $\mathbf{u}^\top \mathbf{v}$ to denote the dot product.
- *Commutative:* $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$.

1.3 Outer Product

$$\mathbf{u}\,\mathbf{v}^\top = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} = \begin{pmatrix} u_1 v_1 & u_1 v_2 & \cdots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \cdots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_n v_1 & u_n v_2 & \cdots & u_n v_n \end{pmatrix}.$$

- The result is an $n \times n$ matrix (assuming both vectors are in $\mathbb{R}^n$).
- *Non-commutative:* $\mathbf{u}\mathbf{v}^\top \neq \mathbf{v}^\top \mathbf{u}$ in general (the latter is a 1×1 matrix, i.e. a scalar).

2 Matrices

Let A be an $m \times n$ matrix and B be an $n \times p$ matrix. Then

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}.$$

2.1 Matrix Addition and Scalar Multiplication

$$A + C = \begin{pmatrix} a_{11} + c_{11} & \cdots & a_{1n} + c_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} + c_{m1} & \cdots & a_{mn} + c_{mn} \end{pmatrix}, \quad \alpha A = \begin{pmatrix} \alpha a_{11} & \cdots & \alpha a_{1n} \\ \vdots & \ddots & \vdots \\ \alpha a_{m1} & \cdots & \alpha a_{mn} \end{pmatrix}.$$

- A and C must be the same dimension for $A + C$ to be defined.
- Commutative: $A + C = C + A$.

2.2 Matrix Multiplication

$$AB = \begin{pmatrix} \sum_{k=1}^{n} a_{1k} b_{k1} & \sum_{k=1}^{n} a_{1k} b_{k2} & \cdots & \sum_{k=1}^{n} a_{1k} b_{kp} \\ \sum_{k=1}^{n} a_{2k} b_{k1} & \sum_{k=1}^{n} a_{2k} b_{k2} & \cdots & \sum_{k=1}^{n} a_{2k} b_{kp} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{mk} b_{k1} & \sum_{k=1}^{n} a_{mk} b_{k2} & \cdots & \sum_{k=1}^{n} a_{mk} b_{kp} \end{pmatrix}.$$

- AB is defined only if the number of columns of A equals the number of rows of B.
- *Not commutative in general*: $AB \neq BA$ (unless A and B have special forms or dimensions).
- *Associative*: $(AB)C = A(BC)$.

2.3 Matrix Transpose

$$A^\top = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}.$$

- $(A^\top)^\top = A$.
- $(AB)^\top = B^\top A^\top$ (reversing the order of multiplication when transposing a product).

2.4 Matrix Inverse (Square Matrices)

If A is an $n \times n$ invertible (non-singular) matrix, A^{-1} is defined by:

$$AA^{-1} = A^{-1}A = I_n,$$

where I_n is the $n \times n$ identity matrix.

- Not every square matrix is invertible (must have nonzero determinant).
- *Inverse of a Product:* $(AB)^{-1} = B^{-1}A^{-1}$ (again, note how the order is reversed).
- $(A^{-1})^\top = (A^\top)^{-1}$ (transpose and inverse also reverse).

3 Matrix–Vector Multiplication

Let A be an $m \times n$ matrix and $\mathbf{x}$ an $n \times 1$ column vector:

$$A\mathbf{x} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{pmatrix}.$$

- The result is an $m \times 1$ vector.
- $(A\mathbf{x})^\top = \mathbf{x}^\top A^\top$.

4 Reordering Rules and Properties

4.1 Transposition Reordering

$$(AB)^\top = B^\top A^\top.$$
$$(\mathbf{u}\mathbf{v}^\top)^\top = \mathbf{v}\mathbf{u}^\top.$$
$$(A^{-1})^\top = (A^\top)^{-1}.$$

These **reverse** the order of multiplication.

4.2 Inverse Reordering

$$(AB)^{-1} = B^{-1}A^{-1}.$$

When you invert a product of two (invertible) matrices, you *reverse* the order of the factors and invert each separately. The same holds in general for any product of multiple matrices:

$$(A_1A_2\cdots A_k)^{-1} \;=\; A_k^{-1}\cdots A_2^{-1}A_1^{-1}.$$

4.3 Associativity of Multiplication

$$(AB)C = A(BC),$$

for conformable dimensions. This *does not* mean $AB = BA$ in general; it only means the way you group them does not matter.

5 Examples of Ordering in Practice

- **Transpose of a product:**
  If $C = AB$, then $C^\top = (AB)^\top = B^\top A^\top$. For instance, if A is $m \times n$ and B is $n \times p$, then $B^\top$ is $p \times n$ and $A^\top$ is $n \times m$, so $B^\top A^\top$ is $p \times m$ (same as $C^\top$).
- **Inverse of a product:**
  If $C = AB$ is invertible, $(AB)^{-1} = B^{-1}A^{-1}$. For instance, if A is $m \times m$ and B is $m \times m$, and both are invertible, then B^{-1} is $m \times m$, A^{-1} is $m \times m$, so $B^{-1}A^{-1}$ is $m \times m$, as needed.

- **Matrix-vector multiplication and reordering:**
  If $\mathbf{x}$ is a vector, then
  $$(AB)\mathbf{x} = A\big(B\mathbf{x}\big).$$
  We cannot change the order to $\mathbf{x}(AB)$ because that makes no sense dimensionally. Associativity only means we can parenthesize in different ways (but still maintain the left-to-right order of A, B, $\mathbf{x}$).

6 Special Matrix Products

6.1 Elementwise (Hadamard) Product

For two matrices A and B of identical dimensions $m \times n$, the **Hadamard product** (elementwise multiplication) is:

$$A \circ B \;=\; \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2n}b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & a_{m2}b_{m2} & \cdots & a_{mn}b_{mn} \end{pmatrix}.$$

Usage in ML: Elementwise products appear in neural-network layers (especially in gating or attention mechanisms) and factorized models (e.g. factorization machines).

6.2 Kronecker Product

For an $m \times n$ matrix A and a $p \times q$ matrix B, the **Kronecker product** $A \otimes B$ is a block matrix of dimension $(mp) \times (nq)$:

$$A \otimes B \;=\; \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}.$$

Usage in ML: Kronecker products are used in covariance structures (e.g. Kronecker-factored approximations in second-order optimization or natural gradient), in multi-task learning, and in tensor-based neural network architectures.

6.3 Khatri–Rao Product

For two matrices A and B each of size $m \times r$, the **Khatri–Rao product** $A \odot B$ is defined as the columnwise Kronecker product:

$$A \odot B \;=\; \big[a^{(1)} \otimes b^{(1)},\; a^{(2)} \otimes b^{(2)},\; \ldots,\; a^{(r)} \otimes b^{(r)}\big]$$

where $a^{(k)}$ and $b^{(k)}$ are the k-th columns of A and B, respectively. Hence $A \odot B$ is an $(m^2) \times r$ matrix.

Usage in ML/Tensor Factorizations: Khatri–Rao products appear in PARAFAC/CANDECOMP decompositions of tensors and certain low-rank structures.

7 Tensor Basics

7.1 Definition and Notation

A **tensor** is a generalization of vectors (1-D) and matrices (2-D) to higher orders. For example, a 3rd-order tensor $\mathcal{X}$ could be thought of as an $I \times J \times K$ array:

$$\mathcal{X} \in \mathbb{R}^{I \times J \times K}.$$

We write elements as $x_{i,j,k}$ for $1 \le i \le I$, $1 \le j \le J$, $1 \le k \le K$.

7.2 Mode-n Multiplication

Mode-n product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $A \in \mathbb{R}^{J \times I_n}$ is denoted $\mathcal{X} \times_n A$ and results in a tensor of size

$$I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N.$$

Roughly, we multiply A by each *mode-n fiber* of $\mathcal{X}$. Tensor decomposition methods (CP, Tucker) are built upon repeated mode-n multiplications.

7.3 Applications in ML

Tensors can represent multi-dimensional data (e.g. temporal, spatial, spectral). Tensor factorization helps in recommender systems, spatiotemporal data modeling, and multi-modal deep learning (e.g. combining text, image, and audio data).

8 Important Matrix Decompositions

8.1 Eigen-Decomposition (Diagonalization)

For a square matrix $A \in \mathbb{R}^{n \times n}$ (assuming it has n independent eigenvectors), one can write:

$$A \;=\; V \Lambda V^{-1},$$

where V is the matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues.

Symmetric matrices ($A = A^\top$): If A is real symmetric, then A can be *orthogonally diagonalized*:

$$A = Q \Lambda Q^\top,$$

where Q is an orthogonal matrix ($Q^{-1} = Q^\top$). This arises frequently in covariance/correlation matrices in ML (PCA).

8.2 Singular Value Decomposition (SVD)

Any $m \times n$ matrix M can be decomposed as

$$M \;=\; U \Sigma V^\top,$$

where

- $U \in \mathbb{R}^{m \times m}$ is orthogonal (or unitary in complex case),
- $V \in \mathbb{R}^{n \times n}$ is orthogonal,
- $\Sigma \in \mathbb{R}^{m \times n}$ is diagonal (in the sense that only the main diagonal may have nonzero values), containing singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$.

Usage in ML: Low-rank approximations (truncating SVD) are used in PCA, collaborative filtering, and compression of large weight matrices in neural nets.

8.3 QR Decomposition

Any $m \times n$ matrix M with full column rank ($n \leq m$) can be factorized as

$$M = QR,$$

where Q is $m \times n$ with orthonormal columns, and R is an $n \times n$ upper triangular matrix.

Usage in ML: QR decompositions often appear in least squares solutions, numerical optimization algorithms, and methods for solving linear systems stably.

8.4 Cholesky Decomposition

A symmetric positive definite matrix A can be decomposed uniquely as

$$A = LL^\top,$$

where L is a lower-triangular matrix with positive diagonal entries.

Usage in ML: Covariance matrices are often approximated via Cholesky for fast sampling (e.g. in Gaussian processes) or solving systems in Bayesian inference.

9 Matrix Calculus and Gradients

Differentiating functions with respect to vectors and matrices is crucial in backpropagation, gradient-based optimization, etc. Let's outline some common rules:

9.1 Gradient w.r.t. a Vector

If $f(\mathbf{x})$ is a scalar function, and $\mathbf{x} \in \mathbb{R}^n$, the gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$ is a vector in $\mathbb{R}^n$:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}.$$

Chain rule example: If $\mathbf{y} = A\mathbf{x}$, then $f(\mathbf{x}) = g(\mathbf{y})$, we have $\nabla_{\mathbf{x}} f(\mathbf{x}) = A^\top \nabla_{\mathbf{y}} g(\mathbf{y})$.

9.2 Gradient w.r.t. a Matrix

If $F(A) \in \mathbb{R}$ for a matrix $A \in \mathbb{R}^{m \times n}$, the gradient is also a matrix of the same shape, where

$$(\nabla_A F)_{ij} = \frac{\partial F}{\partial a_{ij}}.$$

A common identity is:

$$\nabla_A \operatorname{tr}(BA) = B^\top,$$

where $\operatorname{tr}(\cdot)$ is the trace operator.

9.3 Backpropagation in Neural Networks

All of the partial derivatives needed in neural networks (weights, biases, etc.) ultimately reduce to these matrix/vector calculus rules combined with the chain rule.

$$\frac{\partial \,\text{Loss}}{\partial W} = \frac{\partial \,\text{Loss}}{\partial \,\text{Output}} \cdot \frac{\partial \,\text{Output}}{\partial W}.$$

10 Practical Examples in Machine Learning

10.1 Low-Rank Factorization for Parameter Compression

For a large weight matrix $W \in \mathbb{R}^{m \times n}$ in a neural network, one can approximate it via SVD truncation:

$$W \approx U \Sigma V^\top,$$

where Σ is kept only for the top r singular values, leading to a rank-r factorization with fewer parameters.

10.2 Kronecker-Factored Approximate Curvature (K-FAC)

In second-order optimization for deep learning, one approximates the Fisher information matrix by a Kronecker product of smaller matrices:

$$F \approx A \otimes B.$$

Then inverting F is simpler because $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$. K-FAC can significantly speed up training.

10.3 Tensor Decompositions in Recommender Systems

For a 3rd-order tensor $\mathcal{X} \in \mathbb{R}^{U \times I \times T}$ (user $\times$ item $\times$ time), a CP (CANDECOMP/PARAFAC) decomposition might approximate:

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{u}_r \circ \mathbf{i}_r \circ \mathbf{t}_r,$$

where each $\mathbf{u}_r \in \mathbb{R}^U$, $\mathbf{i}_r \in \mathbb{R}^I$, $\mathbf{t}_r \in \mathbb{R}^T$. Learning $\mathbf{u}_r$, $\mathbf{i}_r$, $\mathbf{t}_r$ is then akin to factorizing multi-way data for predictions.

11 Additional Reordering Secrets

11.1 Trace Manipulations

The trace operator has many convenient properties:

$$\operatorname{tr}(ABC) = \operatorname{tr}(CAB) = \operatorname{tr}(BCA),$$

$$\operatorname{tr}(A^\top) = \operatorname{tr}(A),$$

$$\operatorname{tr}(\alpha A) = \alpha \operatorname{tr}(A).$$

Such properties help in simplifying expressions in ML proofs (e.g. expansions of loss functions, gradient derivations).

11.2 Reshape and Vectorization

$\mathbf{vec}(A)$ stacks the columns of A into a single long column vector. Useful identity:

$$\operatorname{vec}(ABC) = (C^\top \otimes A) \operatorname{vec}(B).$$

This is key in advanced backprop computations and in certain tensor expansions.

12 Advanced Machine Learning Applications

In this section, we build upon the linear algebra operations and reordering properties to highlight more sophisticated applications in modern Machine Learning. The focus is on using matrix and tensor operations efficiently and in novel ways to solve complex predictive tasks.

12.1 Matrix Reordering and Efficient Computation

Large-scale Machine Learning models often rely on high-dimensional matrix multiplications. Strategic reordering (e.g., grouping operations via associativity) can improve computational efficiency and memory usage:

- **Batch Multiplication:** If we have a mini-batch of input vectors $\{\mathbf{x}_i\}$ and a weight matrix W, we can write the batched operation as

$$(W\mathbf{X}) = W \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_b \\ | & | & & | \end{pmatrix},$$

  where $\mathbf{X}$ is an $n \times b$ matrix (each column is an input). Reordering via associativity (grouping multiplications for matrix–matrix multiplication) can yield faster GPU implementations than multiple separate vector–matrix multiplications.

- **Transformer Blocks:** In self-attention mechanisms, we often compute $QK^\top$, where Q and K are query/key matrices of dimension (batch $\times$ sequence length $\times$ d). Certain frameworks rearrange these tensors for GPU-friendliness, taking advantage of the fact that

$$(QK^\top)^\top = KQ^\top$$

  so if we need $(QK^\top)^\top$ for some computation, we can store $KQ^\top$ directly, saving one transpose operation.

12.2 Automating Gradient Computation via Matrix Calculus

Automatic differentiation (AD) tools rely heavily on the chain rule in matrix form. The reordering rules are implicitly used to ensure that each gradient is computed in the correct order:

- **Parameter Updates in Networks:** If $\mathbf{h} = \sigma(W\mathbf{x} + \mathbf{b})$ is a hidden layer (with activation σ), then

$$\frac{\partial \mathbf{h}}{\partial W} = \frac{\partial \sigma(\cdot)}{\partial (W\mathbf{x} + \mathbf{b})} \otimes \mathbf{x}^\top,$$

  which arises from applying the chain rule and vectorizing $\mathbf{x}$. The final partial derivative is then rearranged by the AD system to match the shape of W in memory.

- **Second-Order Methods:** When using approximate second-order updates (e.g., K-FAC), we keep

$$(AB)^{-1} = B^{-1}A^{-1},$$

  in mind to more efficiently invert large Kronecker-factored matrices. This property drastically reduces computation compared to inverting the full Fisher information matrix directly.

12.3 Advanced Tensor Operations in Deep Learning

Beyond matrix multiplication, higher-order tensors appear in various deep learning contexts:

- **Convolution as a Matrix Multiplication:** A convolutional layer can be "unfolded" (or im2col-transformed) into a standard matrix multiplication $W \cdot \tilde{\mathbf{X}}$, where $\tilde{\mathbf{X}}$ is a matrix containing patches extracted from images. Modern libraries may reorder dimensions internally to speed up these operations and facilitate vectorization on GPUs.

- **Attention Weights in Transformers:** The 3D tensor for multi-head attention can be reordered via $(\text{batch}, \text{heads}, \text{seq}, d)$ or $(\text{batch}, \text{seq}, \text{heads}, d)$, etc. Reordering ensures that attention weight calculations (e.g. $QK^\top$ and subsequent softmax) are computed efficiently and remain memory-coalesced.

- **Tensor Factorizations in Multi-Modal Learning:** Extending beyond matrices, if a dataset has multiple modalities (e.g. image, text, and audio), each sample can be viewed as a higher-order tensor. Operations like *Khatri–Rao* or *Kronecker* products may be used to integrate or "merge" these modalities. The reordering properties become crucial to avoid repeated overhead in factorized multi-modal architectures.

12.4 Practical Tips for ML Engineers

- **Benchmark Reorderings:** Even though $(AB)C = A(BC)$, different GPU libraries might run faster with a particular parenthesization. When matrices are large, profiling the order of multiplication is often worthwhile.
- **Keep Track of Batch Dimensions:** In many deep learning frameworks, the first dimension is the batch size. Be mindful of whether the library is optimized for (batch $\times \dots$) or ($\dots \times$ batch) ordering. Correct dimension reordering can yield surprising speed-ups in training and inference.
- **Careful with Transposes:** Repeatedly transposing large matrices or tensors can degrade performance. Try to restructure your graph or code so that transposes happen minimally, possibly by applying known reorder rules $(AB)^\top = B^\top A^\top$ to reduce the total number of transpose operations.
- **Matrix Inversions vs. Factorizations:** Inverting a matrix A directly can be costly. Factorizations such as QR, LU, or Cholesky are usually more stable and faster for solving linear systems $A\mathbf{x} = \mathbf{b}$. This is particularly important in Bayesian ML (e.g. Gaussian Processes), where large covariance matrices must be dealt with frequently.