Linear Algebra

Philipp W.

April 6, 2025

Contents

1 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).

2 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

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

2.1 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:* $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$.

2.2 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:* $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$.

2.3 Outer Product

$$\mathbf{u}\,\mathbf{v}^\top = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} = \begin{pmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_nv_1 & u_nv_2 & \cdots & u_nv_n \end{pmatrix}.$$

- 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).

3 Matrices

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

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

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

3.1 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: $A + C = C + A$.

3.2 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*: $(AB)C = A(BC)$.

3.3 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.
- $(A\mathbf{x})^\top = \mathbf{x}^\top A^\top$.

3.4 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).

3.5 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).

$$\textbf{Inverse of a 2x2 matrix:} \quad A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

$$\textbf{General formula via adjugate:} \quad A^{-1} = \frac{1}{\det(A)} \operatorname{adj}(A),$$

where $\operatorname{adj}(A)$ is the *adjugate* (or classical adjoint) of A, obtained by taking the transpose of the cofactor matrix of A.

3.6 Special Matrices and Common Properties

Below are several special classes of matrices frequently encountered in linear algebra and applications such as optimization, statistics, and machine learning.

3.6.1 Identity Matrix

The *identity matrix* I_n is an $n \times n$ matrix with 1's on the main diagonal and 0's elsewhere:

$$I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

- For any $n \times n$ matrix A, $AI_n = I_nA = A$.
- The identity matrix is its own inverse: $I_n^{-1} = I_n$.

3.6.2 All-Ones (Unit) Matrix

Sometimes referred to as the *unit matrix of ones* (not to be confused with the identity matrix), the $m \times n$ all-ones matrix is denoted by $J_{m\times n}$ or simply J when dimensions are clear:

$$J_{m\times n} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}_{m\times n}.$$

- $J_{m\times n}$ has every entry equal to 1.
- When $m = n$, $J_{n\times n}$ can be an eigenvector/eigenvalue example: one eigenvalue is n (with eigenvector $\mathbf{1} = (1, 1, \ldots, 1)^\top$), and the others are 0.

3.6.3 Diagonal Matrix

A matrix $D \in \mathbb{R}^{n\times n}$ is called *diagonal* if $d_{ij} = 0$ for all $i \neq j$:

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix}.$$

- Diagonal matrices commute: $D_1D_2 = D_2D_1$ because multiplication is elementwise for the diagonal entries.
- Inverses and powers of a diagonal matrix are easy to compute (assuming diagonal entries are nonzero).

3.6.4 Symmetric and Hermitian Matrices

A matrix $A \in \mathbb{R}^{n\times n}$ is *symmetric* if $A = A^\top$. Over the complex field, A is *Hermitian* if $A = A^*$ (the conjugate transpose).

- Symmetric real matrices have real eigenvalues and can be diagonalized by an orthogonal matrix (spectral theorem).
- Covariance matrices in statistics are by definition symmetric (and positive semidefinite).

3.6.5 Positive (Semi)Definite Matrices

A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is:

- *Positive semidefinite (PSD)* if for all $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^\top A\mathbf{x} \geq 0$.
- *Positive definite (PD)* if for all $\mathbf{x} \neq \mathbf{0}$, $\mathbf{x}^\top A\mathbf{x} > 0$.

Covariance matrices are PSD, and many optimization algorithms exploit PD/PSD structures (e.g. Hessians, kernel matrices).

3.6.6 Orthogonal (or Orthonormal) Matrices

An *orthogonal matrix* $Q \in \mathbb{R}^{n\times n}$ satisfies $Q^\top Q = QQ^\top = I_n$. In complex spaces, the analogous property is for *unitary* matrices U where $U^*U = I_n$.

- Orthogonal matrices preserve vector norms: $\|Q\mathbf{x}\|_2 = \|\mathbf{x}\|_2$.
- The inverse of an orthogonal matrix is its transpose ($Q^{-1} = Q^\top$).

3.6.7 Trace of a Matrix

The *trace* of a square matrix $A \in \mathbb{R}^{n\times n}$ is the sum of its diagonal elements:

$$\operatorname{tr}(A) = \sum_{i=1}^{n} a_{ii}.$$

– The trace is linear: $\operatorname{tr}(A+B) = \operatorname{tr}(A) + \operatorname{tr}(B)$ and $\operatorname{tr}(\alpha A) = \alpha \operatorname{tr}(A)$ for scalar α.

– Cyclic property: $\operatorname{tr}(ABC) = \operatorname{tr}(CAB) = \operatorname{tr}(BCA)$, which is often used to reorder matrix products inside traces.

3.6.8 Jacobian Matrix

If $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ is a vector-valued function $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))^\top$, then the *Jacobian matrix* $J(\mathbf{f})(\mathbf{x})$ is an $m \times n$ matrix whose (i,j)th entry is $\partial f_i / \partial x_j$. Symbolically,

$$J(\mathbf{f})(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}.$$

3.6.9 Hessian Matrix

If $f : \mathbb{R}^n \to \mathbb{R}$ is a scalar-valued function, the *Hessian matrix* $H_f(\mathbf{x})$ is the $n \times n$ matrix of second partial derivatives:

$$H_f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}.$$

Under suitable smoothness conditions (Schwarz's theorem), mixed partials are equal, so $H_f(\mathbf{x})$ is symmetric.

3.6.10 Covariance Matrix

A covariance matrix $C \in \mathbb{R}^{n\times n}$ is defined for a random vector $\mathbf{X} \in \mathbb{R}^n$ by

$$C = \mathbb{E}\big[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top\big].$$

– C is always symmetric positive semidefinite.

– If all eigenvalues of C are strictly positive, C is positive definite.

– Frequently appears in statistics, Gaussian distributions, and kernel methods in ML.

3.7 Toeplitz Matrices

A *Toeplitz matrix* has constant diagonals:

$$T = \begin{pmatrix} t_0 & t_{-1} & t_{-2} & \cdots & t_{-(n-1)} \\ t_1 & t_0 & t_{-1} & \cdots & t_{-(n-2)} \\ t_2 & t_1 & t_0 & \cdots & t_{-(n-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{n-1} & t_{n-2} & t_{n-3} & \cdots & t_0 \end{pmatrix}$$

– Arise in signal processing and time series analysis

– Can be multiplied by a vector in $O(n \log n)$ time using FFT

3.8 Circulant Matrices

A *circulant matrix* is a special Toeplitz matrix where each row is a cyclic shift of the previous row:

$$C = \begin{pmatrix} c_0 & c_{n-1} & c_{n-2} & \cdots & c_1 \\ c_1 & c_0 & c_{n-1} & \cdots & c_2 \\ c_2 & c_1 & c_0 & \cdots & c_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n-1} & c_{n-2} & c_{n-3} & \cdots & c_0 \end{pmatrix}$$

- Diagonalized by the DFT matrix
- Used in circular convolution

3.9 Vandermonde Matrices

A *Vandermonde matrix* has geometric progression in its columns:

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix}$$

- Determinant: $\det(V) = \prod_{1 \le i < j \le n}(x_j - x_i)$
- Used in polynomial interpolation

3.10 Block Matrices

A *block matrix* is partitioned into submatrices:

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

- Block matrix multiplication follows the same rules as regular matrix multiplication
- Schur complement: $M/A = D - CA^{-1}B$ (if A is invertible)
- Determinant formula: $\det(M) = \det(A)\det(M/A)$

3.11 Hankel Matrices

A *Hankel matrix* has constant anti-diagonals:

$$H = \begin{pmatrix} h_0 & h_1 & h_2 & \cdots & h_{n-1} \\ h_1 & h_2 & h_3 & \cdots & h_n \\ h_2 & h_3 & h_4 & \cdots & h_{n+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{n-1} & h_n & h_{n+1} & \cdots & h_{2n-2} \end{pmatrix}$$

- Used in system identification
- Related to Toeplitz matrices via permutation

4 Tensors

4.1 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 $1 \leq i \leq I$, $1 \leq j \leq J$, $1 \leq k \leq K$.

4.2 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.

4.3 Applications in ML

A tensor is a generalization of vectors and matrices for N-dimensional data and core operand in modern DL frameworks such as PyTorch.

5 Special Operations and Rules

Our operands are vectors, matrices, or general tensors that are related to, can map or contain themselves values. Maps and relations defined on our operands can be different to usual multiplication and addition for scalar operands, which is a common source of error. In the following I provide an overview of noteworthy operation rules.

5.1 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.

5.2 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:

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

5.3 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).

5.4 Reshape and Vectorization

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.

5.5 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.

5.6 Examples of Ordering

- **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, $\mathbf{x}$).

5.7 Special Matrix Products

5.8 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).

5.9 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.

5.10 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.

6 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:

6.1 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 $\nabla_{\mathbf{x}} f(\mathbf{x}) = A^\top \nabla_{\mathbf{y}} g(\mathbf{y})$.

6.2 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.

6.3 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.

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

7 Matrix Decompositions

Matrix (or factorization) decompositions are fundamental tools in linear algebra. They allow us to rewrite matrices in ways that reveal their key properties, such as rank, eigenvalues, or condition numbers. Many of these decompositions have direct applications in machine learning, numerical methods, and data analysis.

7.1 Preliminaries and Definitions

- **Eigenvalues and Eigenvectors:** For a square matrix $A \in \mathbb{R}^{n\times n}$, a scalar $\lambda \in \mathbb{R}$ (or $\mathbb{C}$ in the complex case) is an *eigenvalue* if there exists a nonzero vector $\mathbf{v}$ (the *eigenvector*) such that

$$A\mathbf{v} = \lambda\mathbf{v}.$$

  The set of all eigenvalues forms the *spectrum* (or *spectral set*) of A.

- **Orthogonal (Orthonormal) Basis:** A collection of vectors $\{\mathbf{q}_1, \dots, \mathbf{q}_n\}$ in $\mathbb{R}^n$ is orthonormal if $\mathbf{q}_i^\top \mathbf{q}_j = \delta_{ij}$ (the Kronecker delta). A matrix Q whose columns form an orthonormal set satisfies $Q^\top Q = I$.

- **Diagonal Matrix:** A matrix Λ is *diagonal* if $\Lambda_{ij} = 0$ whenever $i \neq j$. When diagonal entries are real and nonnegative, Λ is sometimes used to list singular values or eigenvalues on the main diagonal.

- **Triangular Matrices:** A matrix L is *lower triangular* if all entries above the main diagonal are zero. Similarly, R (or U) is *upper triangular* if all entries below the main diagonal are zero.

In what follows, we assume basic familiarity with determinants, rank, and vector spaces. Each decomposition has its own requirements (such as symmetry or invertibility).

7.2 Eigen-Decomposition (Diagonalization)

[Eigen-Decomposition Theorem] Let $A \in \mathbb{R}^{n\times n}$ be a square matrix with *n linearly independent* eigenvectors. Then A can be decomposed as

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

where

- V is the $n \times n$ *eigenvector matrix*, whose columns are the eigenvectors of A,
- Λ is the $n \times n$ *diagonal matrix* of eigenvalues (each eigenvalue λ_i appears on the diagonal).

Suppose

$$A = \begin{pmatrix} 4 & 1 \\ 0 & 3 \end{pmatrix}.$$

The eigenvalues are $\lambda_1 = 4, \lambda_2 = 3$. Corresponding eigenvectors might be $\mathbf{v}_1 = (1,0)^\top$ and $\mathbf{v}_2 = (1,1)^\top$. Then

$$V = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \Lambda = \begin{pmatrix} 4 & 0 \\ 0 & 3 \end{pmatrix}, \quad A = V\Lambda V^{-1}.$$

Symmetric matrices ($A = A^\top$): If A is *real symmetric*, the eigen-decomposition simplifies to an *orthogonal diagonalization*:

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

where Q is an orthogonal matrix ($Q^\top Q = I$) and Λ is diagonal with real eigenvalues. This is particularly important in machine learning for covariance/correlation matrices (e.g. PCA).

7.3 Singular Value Decomposition (SVD)

[Singular Value Decomposition] Any $m \times n$ matrix M (real or complex) can be decomposed as

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

where

- $U \in \mathbb{R}^{m\times m}$ is orthogonal (i.e. $U^\top U = I_m$),
- $V \in \mathbb{R}^{n\times n}$ is orthogonal (i.e. $V^\top V = I_n$),
- $\Sigma \in \mathbb{R}^{m\times n}$ is *diagonal* in the sense that only the entries along the main diagonal can be nonzero (these entries are the *singular values*).

Usage in ML:

- *Dimensionality Reduction (PCA):* By taking the top r singular values (and corresponding vectors), we obtain a best rank-r approximation to M.
- *Collaborative Filtering:* Large user–item matrices can be approximated by low-rank decompositions.
- *Model Compression:* Weight matrices in neural networks can sometimes be truncated to reduce the number of parameters.

7.4 QR Decomposition

[QR Factorization] If $M \in \mathbb{R}^{m\times n}$ has full column rank (i.e. rank $n \le m$), then M can be written as

$$M = QR,$$

where

- $Q \in \mathbb{R}^{m\times n}$ has orthonormal columns, and
- $R \in \mathbb{R}^{n\times n}$ is upper triangular.

Usage in ML:

- *Least Squares*: Solving $M\mathbf{x} \approx \mathbf{b}$ can be done stably via Q and R.
- *Numerical Optimization*: Many iterative methods for linear or non-linear problems rely on QR for improved numerical stability.

7.5 Cholesky Decomposition

[Cholesky Factorization] A *symmetric positive definite* matrix $A \in \mathbb{R}^{n\times n}$ can be uniquely decomposed as

$$A = LL^\top,$$

where L is lower-triangular with strictly positive diagonal entries. **Usage in ML:**

- *Covariance Matrices*: Cholesky is commonly used to invert or factor covariance matrices in Gaussian processes and Bayesian inference.
- *Sampling*: To sample from $\mathcal{N}(\mathbf{0}, \Sigma)$, we can set $\Sigma = LL^\top$ and let $\mathbf{z} = L\mathbf{x}$ where $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, I)$.

7.6 LU (or LU) Decomposition

[LU Decomposition] If $M \in \mathbb{R}^{n\times n}$ is a square matrix that can be reduced to an upper-triangular form without row swapping, then there exists a decomposition

$$M = LU,$$

where L is lower triangular (with 1's on the diagonal, in one common convention) and U is upper triangular.

- If row interchanges are needed, one introduces a permutation matrix P to get $PM = LU$.
- *Usage:* LU factorization is another approach to solving systems $M\mathbf{x} = \mathbf{b}$ or computing determinants.

7.7 Schur Decomposition

[Schur Decomposition] Any square matrix $A \in \mathbb{C}^{n\times n}$ can be written as

$$A = QTQ^*,$$

where Q is a unitary matrix ($Q^*Q = I$) and T is an upper triangular matrix. The diagonal entries of T are the eigenvalues of A. This result is often used as an intermediate step toward the Jordan normal form or for proving other matrix decompositions.

7.8 Polar Decomposition

Any invertible matrix $A \in \mathbb{R}^{n\times n}$ (or $\mathbb{C}^{n\times n}$) can be written as

$$A = QH,$$

where Q is orthogonal (or unitary) and H is symmetric positive definite (Hermitian positive definite in the complex case).

- *Usage:* The polar decomposition is analogous to converting a complex number z into $|z|e^{i\theta}$. In ML, it sometimes appears in manifold optimization and shape analysis.

7.9 Additional Examples

[2x2 Cholesky] A simple 2×2 positive definite matrix:

$$A = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}.$$

Its Cholesky decomposition is

$$L = \begin{pmatrix} 2 & 0 \\ 1 & \sqrt{2} \end{pmatrix}, \quad LL^\top = A.$$

[SVD of a rank-1 matrix] Let

$$M = \begin{pmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \end{pmatrix}.$$

This is clearly rank-1 (all rows are multiples of $(2, 2)$). An SVD reveals a single nonzero singular value:

$$\Sigma = \begin{pmatrix} \sqrt{12} & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad U, V \text{ suitably chosen orthogonal matrices.}$$

[LU Example]

$$M = \begin{pmatrix} 2 & 4 & 2 \\ 4 & 8 & 6 \\ 2 & 6 & 9 \end{pmatrix}.$$

One can find

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & \frac{1}{2} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 2 & 4 & 2 \\ 0 & 0 & 2 \\ 0 & 0 & 4 \end{pmatrix}$$

such that $M = LU$.
In summary, each decomposition transforms a matrix into a more insightful or more convenient form, depending on the underlying structure (symmetry, rank, positivity, etc.). These factorizations are powerful both for theoretical analysis (e.g. proofs of eigenvalue properties, spectral theorems) and for practical applications (efficient numerical algorithms, statistical modeling, and advanced optimization in machine learning).

8 Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors reveal the intrinsic directions in which a matrix acts by simple scaling rather than by more complicated transformations (like rotation or shear). They are central to much of modern linear algebra and its applications.

8.1 Definition and Properties

[Eigenvalue, Eigenvector] Let $A \in \mathbb{R}^{n\times n}$ be a square matrix. A nonzero vector $\mathbf{v} \in \mathbb{R}^n$ is called an *eigenvector* of A if there exists a scalar $\lambda \in \mathbb{R}$ (called an *eigenvalue*) such that

$$A\mathbf{v} = \lambda\mathbf{v}.$$

- The *spectrum* of A is the set of all eigenvalues of A.
- For each eigenvalue λ, any scalar multiple of an eigenvector is also an eigenvector.
- The *geometric multiplicity* of λ is the dimension of the eigenspace $\{\mathbf{v} : A\mathbf{v} = \lambda\mathbf{v}\}$.
- The *algebraic multiplicity* of λ is its multiplicity as a root of the characteristic polynomial (below).

[Running Example] Consider the matrix

$$A = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}.$$

This matrix is 2×2 and upper-triangular, making it relatively straightforward to analyze:

- By inspection, the diagonal entries 2 and 3 are the eigenvalues.
- We will find the eigenvectors, characteristic polynomial, and demonstrate diagonalization step-by-step in the subsequent subsections.

8.2 Characteristic Polynomial

The eigenvalues of A are precisely the roots of its *characteristic polynomial*:

$$p_A(\lambda) = \det\bigl(A - \lambda I_n\bigr).$$

[Characteristic Polynomial of the Running Example] Using A from Example 8.1, we compute

$$A - \lambda I_2 = \begin{pmatrix} 2-\lambda & 1 \\ 0 & 3-\lambda \end{pmatrix}.$$

Hence,

$$p_A(\lambda) = \det\begin{pmatrix} 2-\lambda & 1 \\ 0 & 3-\lambda \end{pmatrix} = (2-\lambda)(3-\lambda).$$

The roots (eigenvalues) are $\lambda_1 = 2$ and $\lambda_2 = 3$.

8.3 Diagonalization

[Diagonalizable Matrix] A matrix A is *diagonalizable* if it can be written as

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

where Λ is a diagonal matrix whose diagonal entries are eigenvalues of A, and the columns of V are the corresponding (linearly independent) eigenvectors.

- A matrix A in $\mathbb{R}^{n\times n}$ is diagonalizable if and only if it has n linearly independent eigenvectors.
- If A is a **symmetric** matrix ($A = A^\top$), it is always diagonalizable *via an orthogonal matrix*: $A = Q\Lambda Q^\top$, where $Q^\top Q = I$. All eigenvalues in this case are real.

[Diagonalizing A from Example 8.1] For $A = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$:

1. *Eigenvalue* $\lambda_1 = 2$:
$$(A - 2I_2) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$
This implies $v_2 = 0$, so an eigenvector is $\mathbf{v}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$.

2. *Eigenvalue* $\lambda_2 = 3$:
$$(A - 3I_2) = \begin{pmatrix} -1 & 1 \\ 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} -1 & 1 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$
This gives $-x + y = 0 \implies y = x$. Thus an eigenvector is $\mathbf{v}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

Collecting these vectors, we set
$$V = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \Lambda = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}.$$
You can verify
$$A = V\Lambda V^{-1}.$$
Thus A is diagonalizable, and Λ places the eigenvalues $2, 3$ on the diagonal.

8.4 Jordan Canonical Form

For matrices that are not diagonalizable, the *Jordan form* provides a nearly diagonal representation. In the complex field (or real field, if extended appropriately), any square matrix A can be written as
$$A = P\,J\,P^{-1},$$
where J is block-diagonal with *Jordan blocks* on the diagonal.

8.5 Applications

- **Stability Analysis:** In dynamical systems $\dot{\mathbf{x}} = A\mathbf{x}$, the signs of the real parts of the eigenvalues determine whether trajectories grow, decay, or oscillate around equilibria.
- **Principal Component Analysis (PCA):** Eigenvalues of the covariance matrix Σ indicate the variance captured by each principal direction; eigenvectors are the principal axes.
- **Graph Theory:** Eigenvalues of the adjacency or Laplacian matrix reveal properties such as connectivity, bipartiteness, and expansion of the graph.
- **Quantum Mechanics:** Observable quantities correspond to Hermitian operators, whose eigenvalues are the possible measurement outcomes.

Eigenvalues and eigenvectors thus underlie many theoretical and practical aspects of linear algebra, including decompositions like *Diagonalization* and *Jordan Normal Form*, both of which play an essential role in simplifying or solving matrix equations and modeling phenomena across scientific disciplines.

9 Norms and Inner Products

9.1 Vector Norms

A *norm* on a vector space V is a function $\|\cdot\| : V \to \mathbb{R}$ satisfying:

- $\|\mathbf{x}\| \geq 0$ and $\|\mathbf{x}\| = 0$ iff $\mathbf{x} = \mathbf{0}$
- $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ for scalar α
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (triangle inequality)

Common vector norms:

- L_1 norm: $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$
- L_2 norm: $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$
- L_∞ norm: $\|\mathbf{x}\|_\infty = \max_i |x_i|$
- L_p norm: $\|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$

9.2 Matrix Norms

Matrix norms extend vector norms and include:

- Frobenius norm: $\|A\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$
- Operator norm: $\|A\|_p = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|_p}{\|\mathbf{x}\|_p}$
- Nuclear norm: $\|A\|_* = \sum_i \sigma_i$ (sum of singular values)

9.3 Inner Products

An *inner product* on V is a function $\langle\cdot,\cdot\rangle : V \times V \to \mathbb{R}$ satisfying:

- $\langle\mathbf{x},\mathbf{x}\rangle \geq 0$ and $\langle\mathbf{x},\mathbf{x}\rangle = 0$ iff $\mathbf{x} = \mathbf{0}$
- $\langle\mathbf{x},\mathbf{y}\rangle = \langle\mathbf{y},\mathbf{x}\rangle$
- $\langle\alpha\mathbf{x} + \beta\mathbf{y},\mathbf{z}\rangle = \alpha\langle\mathbf{x},\mathbf{z}\rangle + \beta\langle\mathbf{y},\mathbf{z}\rangle$

9.4 Orthogonality

- Vectors $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal* if $\langle\mathbf{x},\mathbf{y}\rangle = 0$
- A set of vectors is *orthonormal* if they are pairwise orthogonal and have unit norm
- The *orthogonal complement* of a subspace W is $W^\perp = \{\mathbf{x} \mid \langle\mathbf{x},\mathbf{w}\rangle = 0 \text{ for all } \mathbf{w} \in W\}$

9.5 Projections

The *orthogonal projection* of $\mathbf{x}$ onto a subspace W is:

$$P_W\mathbf{x} = \sum_{i=1}^k \langle\mathbf{x},\mathbf{w}_i\rangle\mathbf{w}_i,$$

where $\{\mathbf{w}_1,\ldots,\mathbf{w}_k\}$ is an orthonormal basis for W.

10 Numerical Linear Algebra

10.1 Condition Numbers

The *condition number* of a matrix A measures how sensitive the solution of $A\mathbf{x} = \mathbf{b}$ is to small changes in A or $\mathbf{b}$. Formally, we define

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|.$$

A large value of $\kappa(A)$ indicates an ill-conditioned problem, meaning that even minor perturbations in A or $\mathbf{b}$ can cause significant changes in the solution $\mathbf{x}$. In the special case of the 2-norm, the condition number $\kappa_2(A)$ can be expressed in terms of the largest and smallest singular values, $\sigma_{\max}$ and $\sigma_{\min}$:

$$\kappa_2(A) = \frac{\sigma_{\max}}{\sigma_{\min}}.$$

10.2 Stability of Algorithms

When we implement numerical algorithms in finite-precision arithmetic, we distinguish between different notions of stability:

- **Backward Stability**: The computed solution is the exact solution of a slightly perturbed version of the original problem. In other words, the algorithm might introduce small changes to A or $\mathbf{b}$, but once those changes are accounted for, the final result is mathematically exact.
- **Forward Stability**: The computed solution itself is close to the exact solution of the original (unperturbed) system.
- **Mixed Stability**: Combines aspects of backward and forward stability, allowing some internal steps to be backward stable while also maintaining forward accuracy where it matters most.

10.3 Floating-Point Arithmetic

All numerical computations must account for the limitations of finite-precision arithmetic. The IEEE 754 standard dictates how real numbers are stored and manipulated, including rules for rounding and handling of overflow/underflow. One key parameter is the *machine epsilon* ϵ, which is the smallest number such that $1 + \epsilon > 1$ in floating-point representation. Because of rounding, small errors can accumulate during matrix operations, sometimes leading to *catastrophic cancellation* in subtraction of nearly equal values. This underscores the importance of algorithmic choices that minimize opportunities for numerical instability.

10.4 Iterative Methods

For many large-scale systems, iterative solvers are preferred over direct methods because they better exploit sparse matrices and can be more memory-efficient.

Stationary Methods. Classical methods such as Jacobi iteration, Gauss–Seidel, and Successive Over-Relaxation (SOR) update the solution vector in place based on the most recent values. For example, the Jacobi method updates each component x_i by isolating it from the rest of the system:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}\Big).$$

Gauss–Seidel refines each component sequentially using updated values immediately, while SOR introduces a relaxation parameter to potentially speed up convergence.

Krylov Subspace Methods. More advanced iterative methods build their search directions within a sequence of subspaces generated by repeated multiplication of the system matrix. Conjugate Gradient (CG) is a prototypical example for symmetric positive-definite systems, while GMRES and BiCGSTAB address more general, possibly nonsymmetric problems. These approaches can yield faster convergence, especially when combined with effective *preconditioning*.

10.5 Preconditioning

Preconditioning aims to transform a linear system $A\mathbf{x} = \mathbf{b}$ into a more tractable form for iterative solvers. One common strategy is to solve

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b},$$

where $M \approx A$ but is easier to invert or factorize. This transformation can significantly reduce the number of iterations needed. Popular preconditioners include incomplete LU (ILU) factorizations, diagonal (Jacobi) scaling, and multigrid methods.

10.6 Parallel Computing Considerations

Finally, large-scale linear algebra computations often run on parallel systems. Designing parallel algorithms requires:

- Distributing data among processing elements,
- Minimizing communication overhead,
- Balancing the computational load,
- Optimizing cache usage.

These considerations help maintain high efficiency and numerical stability at scale.

11 Computational Complexity

11.1 Matrix Multiplication

11.1.1 Classical Algorithm

The standard matrix multiplication algorithm for $n \times n$ matrices has complexity $O(n^3)$:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

11.1.2 Strassen's Algorithm

Strassen's algorithm reduces complexity to $O(n^{\log_2 7}) \approx O(n^{2.81})$:

- Divides matrices into 2x2 blocks
- Uses 7 multiplications instead of 8
- Recursively applies the same strategy

11.1.3 Coppersmith-Winograd Algorithm

Theoretical complexity of $O(n^{2.376})$, but not practical for typical matrix sizes.

11.2 Fast Fourier Transform

- Complexity: $O(n \log n)$ for n-point DFT
- Applications:
  - Polynomial multiplication
  - Convolution
  - Signal processing

11.3 Sparse Matrix Operations

- Storage formats:
  - COO (Coordinate format)
  - CSR (Compressed Sparse Row)
  - CSC (Compressed Sparse Column)
- Matrix-vector multiplication: $O(\text{nnz})$ where nnz is number of nonzeros
- Sparse matrix multiplication: $O(\text{flops})$ where flops is number of floating-point operations

11.4 Parallel Computing

11.4.1 Data Distribution

- Block distribution
- Cyclic distribution
- Block-cyclic distribution

11.4.2 Communication Patterns

- Point-to-point communication
- Collective operations (broadcast, reduce, scatter, gather)
- All-to-all communication

11.5 Cache Optimization

- Blocking/tiling for matrix multiplication
- Cache-oblivious algorithms
- Memory hierarchy considerations

11.6 GPU Computing

- CUDA programming model
- Memory coalescing
- Warp-level parallelism
- Shared memory optimization

11.7 Complexity Classes

- P: Problems solvable in polynomial time
- NP: Problems verifiable in polynomial time
- BPP: Problems solvable by probabilistic algorithms
- NC: Problems solvable in polylogarithmic time with polynomial processors

12 Practical Examples in Machine Learning

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 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.

12.2 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.

12.3 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.

12.4 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.

12.5 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.

12.6 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.

- **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 $(\text{batch} \times \dots)$ or $(\dots \times \text{batch})$ 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.