# Mathematics of Transformer

Xianbiao Qi

Intellifusion Inc.

November 20, 2024

# Mathematical foundations of deep learning

1. Linear Algebra [22, 1] → Matrix Analysis [10, 30] →
   Numerical Linear Algebra [24]
   Random Matrix Theory [21]
   Matrix Calculus [16]
2. Calculus → Vector Calculus → Matrix Calculus [16]
3. Probability [3] → Stochastic Process → High-dimensional Probability [26]
   SDE
4. ODE → PDE → SDE [4, 8]
5. Information Theory [28]
6. Convex Optimization [14], Numerical Optimization [15]
7. Mathematics of Machine Learning [6], Probabilistic Machine Learning [13]

# Contents

# Part 1: Linear Algebra
# [23, 22, 5, 10, 11, 26, 14, 15, 16, 7, 6]

# Notation

Definition:

$$\boldsymbol{x} \in \mathbb{R}^d$$

$$\boldsymbol{X} \in \mathbb{R}^{d \times n}$$

$$\boldsymbol{w} \in \mathbb{R}^d$$

$$\boldsymbol{W} \in \mathbb{R}^{d_1 \times d}$$

Simple cases:

$$y = \boldsymbol{w}^\top \boldsymbol{x}, \boldsymbol{y} = (\boldsymbol{W}_1 \boldsymbol{x}), \boldsymbol{Y} = (\boldsymbol{W}_q \boldsymbol{X})^\top (\boldsymbol{W}_k \boldsymbol{X})$$

# Four key problems in Linear Algebra

- Linear equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

- Least square

$$\min \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$$

- Eigenvalue decomposition

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{-1}$$

- Singular value decomposition (SVD)

$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^{\top}$$

# View a matrix in 4 ways

## 1 Viewing a Matrix – 4 Ways

A matrix $(m \times n)$ can be viewed as 1 matrix, $mn$ numbers, $n$ columns and $m$ rows.

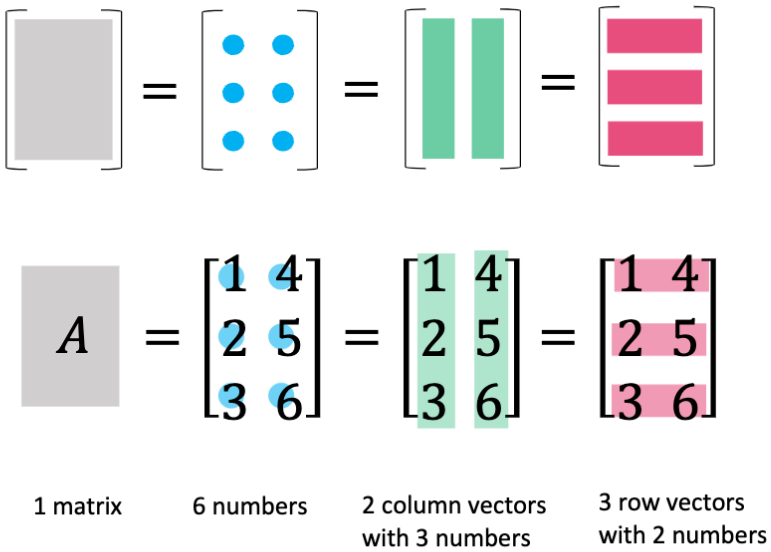
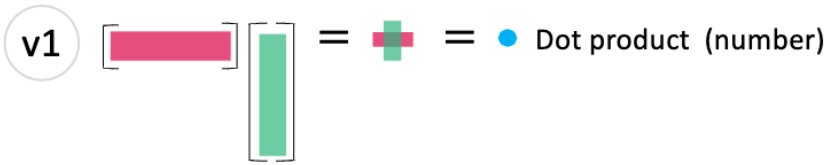


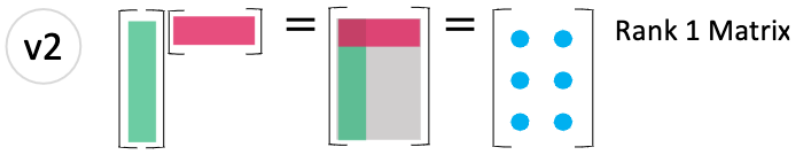

Figure 1: Viewing a Matrix in 4 Ways

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} = \begin{bmatrix} | & | \\ \boldsymbol{a_1} & \boldsymbol{a_2} \\ | & | \end{bmatrix} = \begin{bmatrix} -\boldsymbol{a_1^*}- \\ -\boldsymbol{a_2^*}- \\ -\boldsymbol{a_3^*}- \end{bmatrix}$$

# View vector-by-vector




Dot product ($\boldsymbol{a} \cdot \boldsymbol{b}$) is expressed as $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}$ in matrix language and yields a number.




$\boldsymbol{ab}^{\mathrm{T}}$ is a matrix ($\boldsymbol{ab}^{\mathrm{T}} = A$). If neither $a, b$ are 0, the result $A$ is a rank 1 matrix.

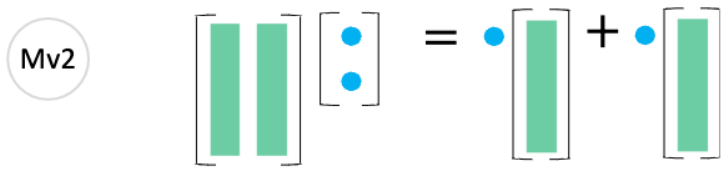
$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = \begin{bmatrix}1\\2\\3\end{bmatrix} \cdot \begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = x_1 + 2x_2 + 3x_3$$

$$\begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}x & y\end{bmatrix} = \begin{bmatrix}x & y\\2x & 2y\\3x & 3y\end{bmatrix}$$

Figure 2: Vector times Vector - (v1), (v2)

# View matrix-by-vector


The row vectors of $A$ are multiplied by a vector $\boldsymbol{x}$ and become the three dot-product elements of $A\boldsymbol{x}$.




The product $A\boldsymbol{x}$ is a linear combination of the column vectors of $A$.

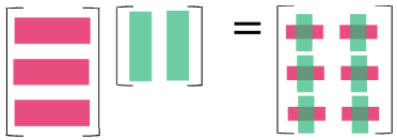
$$A\boldsymbol{x} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} (x_1 + 2x_2) \\ (3x_1 + 4x_2) \\ (5x_1 + 6x_2) \end{bmatrix}$$

$$A\boldsymbol{x} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$$

Figure 3: Matrix times Vector - (Mv1), (Mv2)

# View matrix-by-matrix

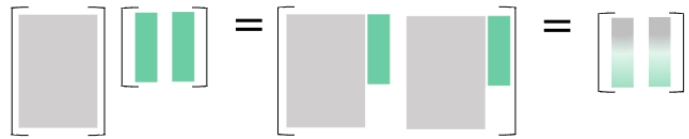
MM 1



Every element becomes a dot product of row vector and column vector.

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}\begin{bmatrix}x_1 & y_1\\ x_2 & y_2\end{bmatrix}=\begin{bmatrix}(x_1+2x_2) & (y_1+2y_2)\\ (3x_1+4x_2) & (3y_1+4y_2)\\ (5x_1+6x_2) & (5y_1+6y_2)\end{bmatrix}$$

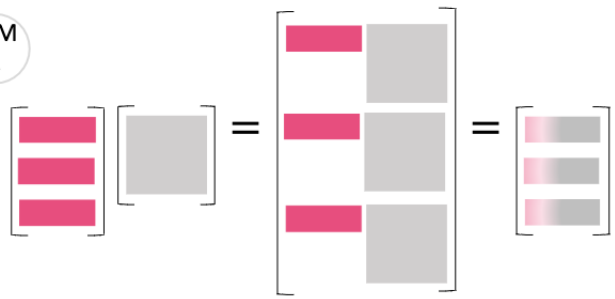
MM 2



$A\boldsymbol{x}$ and $A\boldsymbol{y}$ are linear combinations of columns of $A$.

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}\begin{bmatrix}x_1 & y_1\\ x_2 & y_2\end{bmatrix}=A[\boldsymbol{x}\quad \boldsymbol{y}]=[A\boldsymbol{x}\quad A\boldsymbol{y}]$$

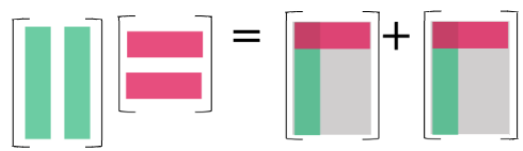
MM 3



The produced rows are linear combinations of rows.

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}\begin{bmatrix}x_1 & y_1\\ x_2 & y_2\end{bmatrix}=\begin{bmatrix}\boldsymbol{a}_1^*\\ \boldsymbol{a}_2^*\\ \boldsymbol{a}_3^*\end{bmatrix}X=\begin{bmatrix}\boldsymbol{a}_1^*X\\ \boldsymbol{a}_2^*X\\ \boldsymbol{a}_3^*X\end{bmatrix}$$

MM 4



Multiplication $AB$ is broken down to a sum of rank 1 matrices.

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}=[\boldsymbol{a}_1\quad \boldsymbol{a}_2]\begin{bmatrix}\boldsymbol{b}_1^*\\ \boldsymbol{b}_2^*\end{bmatrix}=\boldsymbol{a}_1\boldsymbol{b}_1^*+\boldsymbol{a}_2\boldsymbol{b}_2^*$$

$$=\begin{bmatrix}1\\ 3\\ 5\end{bmatrix}[b_{11}\quad b_{12}]+\begin{bmatrix}2\\ 4\\ 6\end{bmatrix}[b_{21}\quad b_{22}]=\begin{bmatrix}b_{11} & b_{12}\\ 3b_{11} & 3b_{12}\\ 5b_{11} & 5b_{12}\end{bmatrix}+\begin{bmatrix}2b_{21} & 2b_{22}\\ 4b_{21} & 4b_{22}\\ 6b_{21} & 6b_{22}\end{bmatrix}$$

Figure 6: Matrix times Matrix - (MM1), (MM2), (MM3), (MM4)

# Matrix decomposition [1]

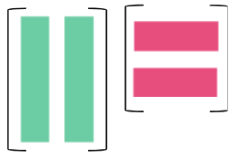
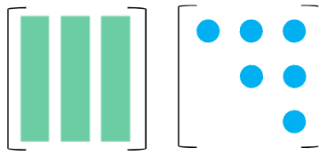
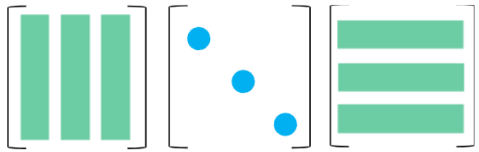
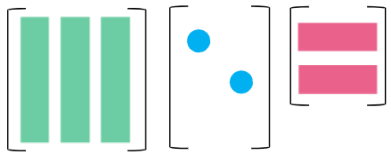
| | | |
|---|---|---|
| $\boldsymbol{A = CR}$ |  | Independent columns in $C$<br>Row echelon form in $R$<br>Leads to column rank = row rank |
| $\boldsymbol{A = LU}$ |  | $LU$ decomposition from<br>Gaussian elimination<br>(Lower triangular)(Upper triangular) |
| $\boldsymbol{A = QR}$ |  | $QR$ decomposition as<br>Gram-Schmidt orthogonalization<br>Orthogonal $Q$ and triangular $R$ |
| $\boldsymbol{S = Q\Lambda Q^{T}}$ |  | Eigenvalue decomposition<br>of a symmetric matrix $S$<br>Eigenvectors in $Q$, eigenvalues in $\Lambda$ |
| $\boldsymbol{A = U\Sigma V^{T}}$ |  | Singular value decomposition<br>of all matrices $A$<br>Singular values in $\Sigma$ |

Table 1: The Five Factorization

[1] https://github.com/kenjihiranabe/The-Art-of-Linear-Algebra

# Matrix world

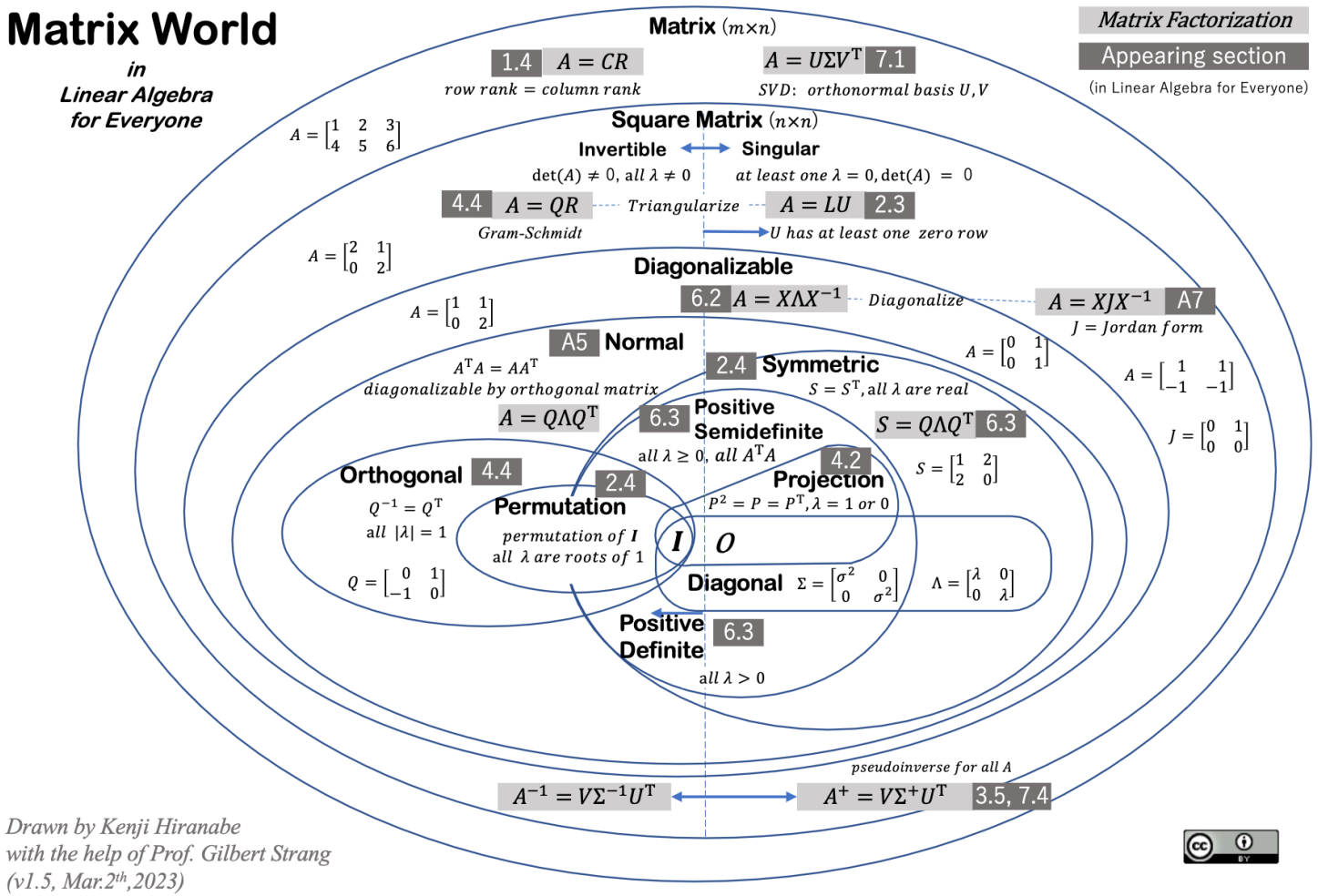



Figure 19: Matrix World

# Rotation matrix

## Definition

A rotation matrix $\boldsymbol{R} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix that preserves distances and orientation:

- $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}$
- $\det(\boldsymbol{R}) = 1$
- $\|\boldsymbol{R}\boldsymbol{x}\| = \|\boldsymbol{x}\|$ for all $\boldsymbol{x}$

## Key Properties

- Inverse equals transpose: $\boldsymbol{R}^{-1} = \boldsymbol{R}^\top$
- Composition of rotations is a rotation
- All eigenvalues have magnitude 1
- Preserves angles between vectors

# 2D rotation matrix

## Counterclockwise Rotation by Angle $\theta$

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

Action on point $(x, y)$:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

# Example: 45-degree rotation

## Rotating by $\theta = \frac{\pi}{4}$

$$\boldsymbol{R}(\pi/4) = \begin{pmatrix} \cos(\pi/4) & -\sin(\pi/4) \\ \sin(\pi/4) & \cos(\pi/4) \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

Example: Rotating point $(1, 0)$:

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

# 3D rotation matrix

## Rotation Around Coordinate Axes

Around x-axis (Roll):

$$\boldsymbol{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

Around y-axis (Pitch):

$$\boldsymbol{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Around z-axis (Yaw):

$$\boldsymbol{R}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

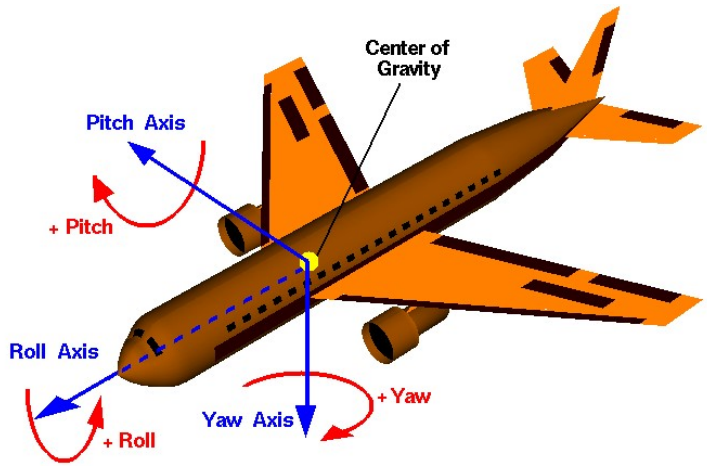



Figure: Roll, Pitch, Yaw.

# Determinant

## Definition

For a square matrix $\boldsymbol{A}$, the determinant $\det(\boldsymbol{A})$ or $|\boldsymbol{A}|$ is a scalar value that provides information about the matrix's invertibility and the scaling factor of the linear transformation it represents.

## Key Properties

For $n \times n$ matrices $\boldsymbol{A}$ and $\boldsymbol{B}$:

- $\det(\boldsymbol{AB}) = \det(\boldsymbol{A})\det(\boldsymbol{B})$
- $\det(\boldsymbol{A}^\top) = \det(\boldsymbol{A})$
- $\det(\boldsymbol{A}^{-1}) = \frac{1}{\det(\boldsymbol{A})}$ if $\boldsymbol{A}$ is invertible
- For triangular matrices: product of diagonal entries

# Computing determinant

For $2 \times 2$ matrix:

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$\det(\boldsymbol{A}) = a_{11}a_{22} - a_{12}a_{21}$$

For $3 \times 3$ matrix:

$$|\boldsymbol{A}| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

Use cofactor expansion

## Geometric Interpretation

The determinant represents:

- Area (2D) or Volume (3D) scaling factor

# Trace

## Definition

For a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, the trace of $\boldsymbol{A}$, denoted $\text{trace}(\boldsymbol{A})$, is the sum of its diagonal elements:

$$\text{trace}(\boldsymbol{A}) = \sum_{i=1}^{n} a_{ii}$$

## Basic Properties

For matrices $\boldsymbol{A}, \boldsymbol{B}$ and scalar $c$:

- Linearity: $\text{trace}(\boldsymbol{A} + \boldsymbol{B}) = \text{trace}(\boldsymbol{A}) + \text{trace}(\boldsymbol{B})$
- Scalar multiplication: $\text{trace}(c\boldsymbol{A}) = c\,\text{trace}(\boldsymbol{A})$
- Transpose: $\text{trace}(\boldsymbol{A}) = \text{trace}(\boldsymbol{A}^\top)$
- Trace of identity: $\text{trace}(\boldsymbol{I}_n) = n$

# Properties of trace

## Cyclic Property

For matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of compatible dimensions:

$$\text{trace}(\boldsymbol{AB}) = \text{trace}(\boldsymbol{BA})$$

More generally:

$$\text{trace}(\boldsymbol{ABP}) = \text{trace}(\boldsymbol{BPA}) = \text{trace}(\boldsymbol{PAB})$$

## Connection to Matrix Operations

- Similar matrices: If $\boldsymbol{B} = \boldsymbol{P}^{-1}\boldsymbol{AP}$, then $\text{trace}(\boldsymbol{A}) = \text{trace}(\boldsymbol{B})$
- For outer product: $\text{trace}(\boldsymbol{x}\boldsymbol{x}^\top) = \boldsymbol{x}^\top\boldsymbol{x}$
- Frobenius norm: $\|\boldsymbol{A}\|_F^2 = \text{trace}(\boldsymbol{A}^\top\boldsymbol{A})$

# Examples

For $2 \times 2$ matrix:

$$\boldsymbol{A} = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix}$$

$$\text{trace}(\boldsymbol{A}) = 3 + 4 = 7$$

For block matrix:

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix}$$

$$\text{trace}(\boldsymbol{B}) = \text{trace}(\boldsymbol{A}_{11}) + \text{trace}(\boldsymbol{A}_{22})$$

# Rank

## Definition

For a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, the rank of $\boldsymbol{A}$, denoted $\text{rank}(\boldsymbol{A})$, can be defined in multiple equivalent ways:

- Dimension of the column space: $\text{rank}(\boldsymbol{A}) = \dim(\text{col}(\boldsymbol{A}))$
- Dimension of the row space: $\text{rank}(\boldsymbol{A}) = \dim(\text{row}(\boldsymbol{A}))$
- Number of linearly independent columns or rows
- Number of non-zero singular values

# Fundamental properties

## Basic Properties

For a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$:

- $0 \leq \text{rank}(\boldsymbol{A}) \leq \min(m, n)$
- $\text{rank}(\boldsymbol{A}) = \text{rank}(\boldsymbol{A}^\top)$
- $\text{rank}(\boldsymbol{AB}) \leq \min(\text{rank}(\boldsymbol{A}), \text{rank}(\boldsymbol{B}))$
- For square matrices: Full rank $\iff$ invertible

# Computing rank

## Methods to Find Rank

- Gaussian elimination
- Compute determinants of submatrices
- SVD decomposition: count non-zero singular values

## Matrix Decompositions

For a matrix $\boldsymbol{A}$ with $\text{rank}(\boldsymbol{A}) = r$:

- Reduced SVD: $\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$ with $r$ singular values
- Full rank factorization: $\boldsymbol{A} = \boldsymbol{F}\boldsymbol{G}$ where $\boldsymbol{F} \in \mathbb{R}^{m\times r}$, $\boldsymbol{G} \in \mathbb{R}^{r\times n}$

# Applications

## System of Linear Equations

For the system $\boldsymbol{Ax} = \boldsymbol{b}$:

- Solution exists $\iff$ $\text{rank}(\boldsymbol{A}) = \text{rank}([\boldsymbol{A}|\boldsymbol{b}])$
- Unique solution $\iff$ $\text{rank}(\boldsymbol{A}) = n$ (full column rank)
- No solution $\iff$ $\text{rank}(\boldsymbol{A}) < \text{rank}([\boldsymbol{A}|\boldsymbol{b}])$

## Applications

- Linear independence analysis
- Matrix invertibility testing
- Image and signal processing (low-rank approximations)
- Data compression and dimensionality reduction

# Examples

Full Rank Matrix:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$\text{rank}(\boldsymbol{A}) = 2$

Rank Deficient Matrix:

$$\boldsymbol{B} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

$\text{rank}(\boldsymbol{B}) = 1$

## Rank and Matrix Properties

- $\text{rank}(\boldsymbol{A} + \boldsymbol{B}) \leq \text{rank}(\boldsymbol{A}) + \text{rank}(\boldsymbol{B})$
- $\text{rank}(\boldsymbol{A}\boldsymbol{A}^\top) = \text{rank}(\boldsymbol{A}^\top \boldsymbol{A}) = \text{rank}(\boldsymbol{A})$

# Matrix inverse

## Definition

For a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, its inverse $\boldsymbol{A}^{-1}$ (if it exists) satisfies:

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$$

## Basic Properties

If $\boldsymbol{A}$ is invertible:

- $(\boldsymbol{A}^{-1})^{-1} = \boldsymbol{A}$
- $(\boldsymbol{A}^{\top})^{-1} = (\boldsymbol{A}^{-1})^{\top}$
- $(c\boldsymbol{A})^{-1} = \frac{1}{c}\boldsymbol{A}^{-1}$ for scalar $c \neq 0$
- $(\boldsymbol{A}\boldsymbol{B})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$ (reverse order!)

# Invertibility conditions

## Equivalent Conditions

For a square matrix $\boldsymbol{A}$, the following are equivalent:

- $\boldsymbol{A}$ is invertible
- $\det(\boldsymbol{A}) \neq 0$
- $\text{rank}(\boldsymbol{A}) = n$
- The columns of $\boldsymbol{A}$ are linearly independent
- The equation $\boldsymbol{Ax} = \boldsymbol{b}$ has a unique solution for all $\boldsymbol{b}$
- 0 is not an eigenvalue of $\boldsymbol{A}$

# Computing inverses

## Methods for Finding $\boldsymbol{A}^{-1}$

1. Adjugate Method (for $2 \times 2$ and $3 \times 3$):

$$\boldsymbol{A}^{-1} = \frac{1}{\det(\boldsymbol{A})} \operatorname{adj}(\boldsymbol{A})$$

2. Gaussian Elimination:

$$[\boldsymbol{A}|\boldsymbol{I}] \rightsquigarrow [\boldsymbol{I}|\boldsymbol{A}^{-1}]$$

3. For $2 \times 2$ matrix:

$$\boldsymbol{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \implies \boldsymbol{A}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

# Special inverse matrix

## Notable Cases

- Orthogonal matrix: $\boldsymbol{A}^{-1} = \boldsymbol{A}^{\top}$
- Diagonal matrix: $\boldsymbol{A}^{-1}$ has reciprocal diagonal entries
- Triangular matrix: $\boldsymbol{A}^{-1}$ has same triangular form
- Symmetric matrix: $\boldsymbol{A}^{-1}$ is also symmetric

## Useful Results

- If $\boldsymbol{A}$ is invertible: $(\boldsymbol{A}^k)^{-1} = (\boldsymbol{A}^{-1})^k$
- Inverse of block diagonal matrix is block diagonal
- For similar matrices: If $\boldsymbol{B} = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}$, then $\boldsymbol{B}^{-1} = \boldsymbol{P}^{-1}\boldsymbol{A}^{-1}\boldsymbol{P}$

# Applications

## Solving Linear Systems

For $\boldsymbol{Ax} = \boldsymbol{b}$ with invertible $\boldsymbol{A}$:

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

# Kronecker product

## Definition

For $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, the Kronecker product $\boldsymbol{A} \otimes \boldsymbol{B}$ is:

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

## Dimensions

- Result is $(mp) \times (nq)$ matrix
- Each $a_{ij}\boldsymbol{B}$ is a $p \times q$ block

# Properties

## Fundamental Properties

For compatible matrices and scalars:

- Bilinearity: $(c\boldsymbol{A}) \otimes \boldsymbol{B} = \boldsymbol{A} \otimes (c\boldsymbol{B}) = c(\boldsymbol{A} \otimes \boldsymbol{B})$
- Associativity: $(\boldsymbol{A} \otimes \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes (\boldsymbol{B} \otimes \boldsymbol{C})$
- Not commutative: $\boldsymbol{A} \otimes \boldsymbol{B} \neq \boldsymbol{B} \otimes \boldsymbol{A}$ in general
- Distributivity: $(\boldsymbol{A} + \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{C} + \boldsymbol{B} \otimes \boldsymbol{C}$

## Transpose Property

$$(\boldsymbol{A} \otimes \boldsymbol{B})^\top = \boldsymbol{A}^\top \otimes \boldsymbol{B}^\top$$

# Important properties

## Matrix Operations

For compatible matrices:

- $(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = (\boldsymbol{AC}) \otimes (\boldsymbol{BD})$
- $(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$ (if both invertible)
- $\det(\boldsymbol{A} \otimes \boldsymbol{B}) = (\det \boldsymbol{A})^n (\det \boldsymbol{B})^m$
- $\operatorname{rank}(\boldsymbol{A} \otimes \boldsymbol{B}) = \operatorname{rank}(\boldsymbol{A}) \operatorname{rank}(\boldsymbol{B})$

## With Identity Matrices

- $\boldsymbol{I}_m \otimes \boldsymbol{I}_n = \boldsymbol{I}_{mn}$
- $(\boldsymbol{I}_m \otimes \boldsymbol{A})(\boldsymbol{I}_m \otimes \boldsymbol{B}) = \boldsymbol{I}_m \otimes (\boldsymbol{AB})$

# Vectorization

## The Vec Operator

For a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\text{vec}(\boldsymbol{A})$ stacks the columns of $\boldsymbol{A}$ into a single vector:

$$\boldsymbol{A} = [\boldsymbol{a}_1 \ \boldsymbol{a}_2 \ \cdots \ \boldsymbol{a}_n] \implies \text{vec}(\boldsymbol{A}) = \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_n \end{bmatrix}$$

where $a_i$ are the columns of $\boldsymbol{A}$

## Dimensions

- Input: $\boldsymbol{A} \in \mathbb{R}^{m \times n}$
- Output: $\text{vec}(\boldsymbol{A}) \in \mathbb{R}^{mn}$

# Properties

## Linear Properties

For matrices $\boldsymbol{A}, \boldsymbol{B}$ and scalar $c$:

- $\text{vec}(\boldsymbol{A} + \boldsymbol{B}) = \text{vec}(\boldsymbol{A}) + \text{vec}(\boldsymbol{B})$
- $\text{vec}(c\boldsymbol{A}) = c\,\text{vec}(\boldsymbol{A})$
- $\text{vec}(\boldsymbol{A}^\top) \neq \text{vec}(\boldsymbol{A})$ in general

## Inner Product Relation

For matrices $\boldsymbol{A}, \boldsymbol{B}$ of same size:

$$\text{trace}(\boldsymbol{A}^\top \boldsymbol{B}) = \text{vec}(\boldsymbol{A})^\top \text{vec}(\boldsymbol{B})$$

# Connection of Kronecker product to Vec operator

## Vec Operator

For matrix $\boldsymbol{A}$, $\text{vec}(\boldsymbol{A})$ stacks columns of $\boldsymbol{A}$ into a vector

## Important Relations

- $\text{vec}(\boldsymbol{ABC}) = (\boldsymbol{C}^\top \boldsymbol{B}^\top \otimes \boldsymbol{I}_m)\,\text{vec}(\boldsymbol{A}) = (\boldsymbol{C}^\top \otimes \boldsymbol{A})\,\text{vec}(\boldsymbol{B}) = (\boldsymbol{I}_l \otimes \boldsymbol{AB})\,\text{vec}(\boldsymbol{C})$
- $\text{vec}(\boldsymbol{AB}) = (\boldsymbol{I} \otimes \boldsymbol{A})\,\text{vec}(\boldsymbol{B}) = (\boldsymbol{B}^\top \otimes \boldsymbol{I})\,\text{vec}(\boldsymbol{A})$
- These relations are crucial in matrix equations and optimization

# Example

## Simple Example

For $\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} 1\boldsymbol{B} & 2\boldsymbol{B} \\ 3\boldsymbol{B} & 4\boldsymbol{B} \end{bmatrix} = \begin{bmatrix} a & b & 2a & 2b \\ c & d & 2c & 2d \\ 3a & 3b & 4a & 4b \\ 3c & 3d & 4c & 4d \end{bmatrix}$$

# Kronecker product and Vec operator

## Fundamental Identity

For compatible matrices $\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{B}$:

$$\operatorname{vec}(\boldsymbol{AXB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$$

## Special Cases

- $\operatorname{vec}(\boldsymbol{AX}) = (\boldsymbol{I} \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$
- $\operatorname{vec}(\boldsymbol{XB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{I}) \operatorname{vec}(\boldsymbol{X})$
- $\operatorname{vec}(\boldsymbol{Ax}) = (\boldsymbol{x}^\top \otimes \boldsymbol{I}) \operatorname{vec}(\boldsymbol{A})$

# Properties of the Vec operator

## Key Properties

For matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{X}$ and vectors $\boldsymbol{a}$, $\boldsymbol{c}$:

$$\operatorname{vec}(\boldsymbol{AXB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$$

$$\operatorname{trace}(\boldsymbol{A}^\top \boldsymbol{B}) = \operatorname{vec}(\boldsymbol{A})^\top \operatorname{vec}(\boldsymbol{B})$$

$$\operatorname{vec}(\boldsymbol{A} + \boldsymbol{B}) = \operatorname{vec}(\boldsymbol{A}) + \operatorname{vec}(\boldsymbol{B})$$

$$\operatorname{vec}(\alpha \boldsymbol{A}) = \alpha \cdot \operatorname{vec}(\boldsymbol{A})$$

$$\boldsymbol{a}^\top \boldsymbol{X} \boldsymbol{B} \boldsymbol{X}^\top \boldsymbol{c} = \operatorname{vec}(\boldsymbol{X})^\top (\boldsymbol{B} \otimes \boldsymbol{c}\boldsymbol{a}^\top) \operatorname{vec}(\boldsymbol{X})$$

# Commutation Matrix

Let $\boldsymbol{A}$ denote the following $3 \times 2$ matrix:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}.$$

$\boldsymbol{A}$ has the following column-major and row-major vectorizations (respectively):

$$\boldsymbol{v}_{\text{col}} = \text{vec}(\boldsymbol{A}) = [1, 2, 3, 4, 5, 6]^\top, \quad \boldsymbol{v}_{\text{row}} = \text{vec}(\boldsymbol{A}^\top) = [1, 4, 2, 5, 3, 6]^\top$$

The associated commutation matrix is

$$\boldsymbol{K} = \boldsymbol{K}^{(3,2)} = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix},$$

# Expectation of linear combinations

Assume $\boldsymbol{X}$ and $\boldsymbol{x}$ to be a matrix and a vector of random variables. Then:

$$E[\boldsymbol{AXB} + \boldsymbol{C}] = \boldsymbol{A}E[\boldsymbol{X}]\boldsymbol{B} + \boldsymbol{C}$$
$$\text{Var}[\boldsymbol{Ax}] = \boldsymbol{A}\text{Var}[\boldsymbol{x}]\boldsymbol{A}^\top$$
$$\text{Cov}[\boldsymbol{Ax}, \boldsymbol{By}] = \boldsymbol{A}\text{Cov}[\boldsymbol{x}, \boldsymbol{y}]\boldsymbol{B}^\top$$

Assume $\boldsymbol{x}$ to be a stochastic vector with mean $\boldsymbol{m}$, then:

$$E[\boldsymbol{Ax} + \boldsymbol{b}] = \boldsymbol{Am} + \boldsymbol{b}$$
$$E[\boldsymbol{Ax}] = \boldsymbol{Am}$$
$$E[\boldsymbol{x} + \boldsymbol{b}] = \boldsymbol{m} + \boldsymbol{b}$$

# Expectation I

Assume $\boldsymbol{x}$ to be a stochastic vector with mean $\boldsymbol{m}$, and covariance $\boldsymbol{M}$. Then:

$$\mathbb{E}[(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{a})(\boldsymbol{B}\boldsymbol{x}+\boldsymbol{b})^\top] = \boldsymbol{A}\boldsymbol{M}\boldsymbol{B}^\top + (\boldsymbol{A}\boldsymbol{m}+\boldsymbol{a})(\boldsymbol{B}\boldsymbol{m}+\boldsymbol{b})^\top$$

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top] = \boldsymbol{M} + \boldsymbol{m}\boldsymbol{m}^\top$$

$$\mathbb{E}[\boldsymbol{x}\boldsymbol{a}^\top\boldsymbol{x}] = (\boldsymbol{M} + \boldsymbol{m}\boldsymbol{m}^\top)\boldsymbol{a}$$

$$\mathbb{E}[\boldsymbol{x}^\top\boldsymbol{a}\boldsymbol{x}^\top] = \boldsymbol{a}^\top(\boldsymbol{M} + \boldsymbol{m}\boldsymbol{m}^\top)$$

$$\mathbb{E}[(\boldsymbol{A}\boldsymbol{x})(\boldsymbol{A}\boldsymbol{x})^\top] = \boldsymbol{A}(\boldsymbol{M} + \boldsymbol{m}\boldsymbol{m}^\top)\boldsymbol{A}^\top$$

$$\mathbb{E}[(\boldsymbol{x}+\boldsymbol{a})(\boldsymbol{x}+\boldsymbol{a})^\top] = \boldsymbol{M} + (\boldsymbol{m}+\boldsymbol{a})(\boldsymbol{m}+\boldsymbol{a})^\top$$

# Expectation II

Trace formulas for the stochastic vector $\boldsymbol{x}$ with mean $\boldsymbol{m}$ and covariance $\boldsymbol{M}$:

$$\mathbb{E}[(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{a})^\top(\boldsymbol{B}\boldsymbol{x}+\boldsymbol{b})] = \text{trace}(\boldsymbol{A}\boldsymbol{M}\boldsymbol{B}^\top) + (\boldsymbol{A}\boldsymbol{m}+\boldsymbol{a})^\top(\boldsymbol{B}\boldsymbol{m}+\boldsymbol{b})$$

$$\mathbb{E}[\boldsymbol{x}^\top\boldsymbol{x}] = \text{trace}(\boldsymbol{M}) + \boldsymbol{m}^\top\boldsymbol{m}$$

$$\mathbb{E}[\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x}] = \text{trace}(\boldsymbol{A}\boldsymbol{M}) + \boldsymbol{m}^\top\boldsymbol{A}\boldsymbol{m}$$

$$\mathbb{E}[(\boldsymbol{A}\boldsymbol{x})^\top(\boldsymbol{A}\boldsymbol{x})] = \text{trace}(\boldsymbol{A}\boldsymbol{M}\boldsymbol{A}^\top) + (\boldsymbol{A}\boldsymbol{m})^\top(\boldsymbol{A}\boldsymbol{m})$$

$$\mathbb{E}[(\boldsymbol{x}+\boldsymbol{a})^\top(\boldsymbol{x}+\boldsymbol{a})] = \text{trace}(\boldsymbol{M}) + (\boldsymbol{m}+\boldsymbol{a})^\top(\boldsymbol{m}+\boldsymbol{a})$$

# Gaussian probability density function

The density of $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{m}, \Sigma)$ is

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m})^\top \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{m})\right]$$

Note that if $\boldsymbol{x}$ is $d$-dimensional, then:

$$\det(2\pi\Sigma) = (2\pi)^d \det(\Sigma)$$

Integration and normalization properties follow from this form.

# Mean and Covariance

First and second moments. Assume $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{m}, \Sigma)$

$$\mathbb{E}(\boldsymbol{x}) = \boldsymbol{m}$$

$$\text{Cov}(\boldsymbol{x}, \boldsymbol{x}) = \text{Var}(\boldsymbol{x}) = \Sigma = \mathbb{E}(\boldsymbol{x}\boldsymbol{x}^\top) - \mathbb{E}(\boldsymbol{x})\mathbb{E}(\boldsymbol{x}^\top) = \mathbb{E}(\boldsymbol{x}\boldsymbol{x}^\top) - \boldsymbol{m}\boldsymbol{m}^\top$$

As for any other distribution, it holds for gaussians that:

$$\mathbb{E}[\boldsymbol{A}\boldsymbol{x}] = \boldsymbol{A}\mathbb{E}[\boldsymbol{x}]$$
$$\text{Var}[\boldsymbol{A}\boldsymbol{x}] = \boldsymbol{A}\text{Var}[\boldsymbol{x}]\boldsymbol{A}^\top$$
$$\text{Cov}[\boldsymbol{A}\boldsymbol{x}, \boldsymbol{B}\boldsymbol{y}] = \boldsymbol{A}\text{Cov}[\boldsymbol{x}, \boldsymbol{y}]\boldsymbol{B}^\top$$

Assume $\boldsymbol{A}$ is $m \times n$ and consider the linear system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

Construct the augmented matrix $\boldsymbol{B} = [\boldsymbol{A}, \boldsymbol{b}]$, then

| Condition | Solution |
|---|---|
| $\text{rank}(\boldsymbol{A}) = \text{rank}(\boldsymbol{B}) = n$ | Unique solution $\boldsymbol{x}$ |
| $\text{rank}(\boldsymbol{A}) = \text{rank}(\boldsymbol{B}) < n$ | Many solutions $\boldsymbol{x}$ |
| $\text{rank}(\boldsymbol{A}) < \text{rank}(\boldsymbol{B})$ | No solutions $\boldsymbol{x}$ |

## Matrix Form

A system of $m$ linear equations in $n$ unknowns:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where:

- $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{x} \in \mathbb{R}^{n}$, $\boldsymbol{b} \in \mathbb{R}^{m}$.

## Augmented Matrix

$$[\boldsymbol{A}|\boldsymbol{b}] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array}\right]$$

# Solution existence

## Fundamental Theorem

For system $\boldsymbol{Ax} = \boldsymbol{b}$:

- Has solution $\iff$ $\text{rank}(\boldsymbol{A}) = \text{rank}([\boldsymbol{A}|\boldsymbol{b}])$
- Has unique solution $\iff$ $\text{rank}(\boldsymbol{A}) = n$
- Has infinitely many solutions $\iff$ $\text{rank}(\boldsymbol{A}) < n$ and consistent
- Has no solution $\iff$ $\text{rank}(\boldsymbol{A}) < \text{rank}([\boldsymbol{A}|\boldsymbol{b}])$

# Solution methods: direct

## Gaussian Elimination

Steps:

1. Forward elimination (to upper triangular)
2. Back substitution
3. Solution verification

## LU Decomposition

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

- Solve $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$ (forward substitution)
- Solve $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$ (back substitution)
- Efficient for multiple right-hand sides

## Objective

Given:

- Data matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$
- Target vector $\boldsymbol{b} \in \mathbb{R}^m$
- Goal: Find $\boldsymbol{x} \in \mathbb{R}^n$ that minimizes:

$$J(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

## Normal Equations

The optimal solution satisfies:

$$\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^\top \boldsymbol{b}$$

# LMS algorithm

## Gradient Descent Approach

The gradient of the cost function:

$$\nabla J(\boldsymbol{x}) = 2\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

Update rule:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mu \nabla J(\boldsymbol{x}_k)$$

where $\mu$ is the learning rate

**Algorithm 1** LMS Algorithm

1: Initialize $\boldsymbol{x}_0$ randomly
2: Choose learning rate $\mu > 0$
3: **while** not converged **do**
4: $\quad \boldsymbol{e}_k = \boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}$ ▷ Compute error
5: $\quad \boldsymbol{g}_k = 2\boldsymbol{A}^\top \boldsymbol{e}_k$ ▷ Compute gradient
6: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mu \boldsymbol{g}_k$ ▷ Update estimate
7: **end while**

# Stochastic LMS

## Online Learning

Process one sample at a time:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \mu(\boldsymbol{b}_i - \boldsymbol{a}_i^\top \boldsymbol{x}_k)\boldsymbol{a}_i$$

where $\boldsymbol{a}_i^\top$ is the $i$-th row of $\boldsymbol{A}$

## Properties

- Lower memory requirements
- Handles streaming data
- Noisy updates but converges in expectation

# 3. Eigenvalue decomposition

## Definition

For a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$:

- A scalar $\lambda \in \mathbb{C}$ is an eigenvalue if:

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}$$

  for some nonzero vector $\boldsymbol{v} \in \mathbb{C}^n$

- $\boldsymbol{v}$ is called an eigenvector corresponding to $\lambda$

## Characteristic Equation

$$\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0$$

This polynomial equation of degree $n$ has $n$ complex roots (counting multiplicity)

# Eigenvalue decomposition

## Definition

A matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is diagonalizable if:

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{-1}$$

where:

- $\boldsymbol{P}$ is the matrix of eigenvectors
- $\boldsymbol{D}$ is a diagonal matrix of eigenvalues
- Columns of $\boldsymbol{P}$ are eigenvectors of $\boldsymbol{A}$

## Key Properties

- $\boldsymbol{A}^k = \boldsymbol{P}\boldsymbol{D}^k\boldsymbol{P}^{-1}$
- $\det(\boldsymbol{A}) = \prod_{i=1}^{n} \lambda_i$
- $\text{trace}(\boldsymbol{A}) = \sum_{i=1}^{n} \lambda_i$

# Example: $2 \times 2$ case

## For Matrix

$$\boldsymbol{A} = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$$

Eigenvalues:

$$\lambda_1 = 4,\ \lambda_2 = 2$$

Eigenvectors:

$$\boldsymbol{v}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix},\ \boldsymbol{v}_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Decomposition:

$$\boldsymbol{A} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}^{-1}$$

# 4. Singular value decomposition (SVD)

$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$$

where, $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}, \boldsymbol{V}\boldsymbol{V}^\top = \boldsymbol{I}$.

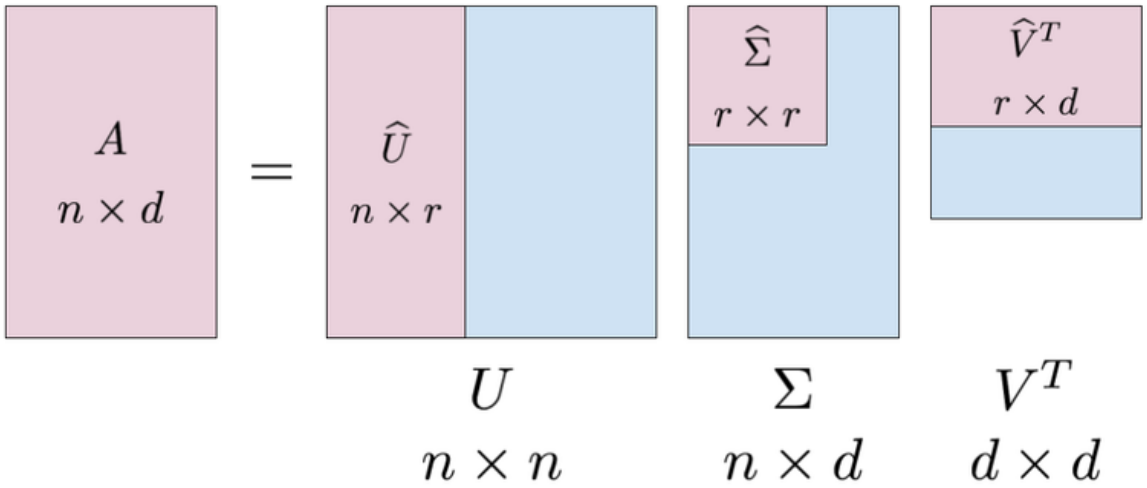



Figure: Illustration[2] of Singular Value Decomposition.

[2] https://www.linkedin.com/pulse/ml-algorithm-singular-value-decomposition-angela-ju/

```python
#Demonstrate code of SVD
import torch

# Create a simple matrix
X = torch.tensor([[1., 2., 3.],
                  [4., 5., 6.]])

# Perform SVD
U, S, V = torch.svd(X)

# Reconstruct the matrix
X_reconstructed = U @ torch.diag(S) @ V.T

print("Original matrix:")
print(X)
print("\nReconstructed matrix:")
print(X_reconstructed)
print("\nReconstruction error:")
print(torch.norm(X - X_reconstructed))
```

```
Original matrix:
tensor([[1., 2., 3.],
        [4., 5., 6.]])

Reconstructed matrix:
tensor([[1.0000, 2.0000, 3.0000],
        [4.0000, 5.0000, 6.0000]])

Reconstruction error:
tensor(1.8961e-06)
```

Figure: Demonstrate code for SVD.

## Definition

For any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, there exists a factorization:

$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$$

where:

- $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ is orthogonal ($\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}_m$)
- $\Sigma \in \mathbb{R}^{m \times n}$ is diagonal with non-negative entries
- $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ is orthogonal ($\boldsymbol{V}^\top \boldsymbol{V} = \boldsymbol{I}_n$)

## Definition

For any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, there exists a factorization:

$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^\top$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{U} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{5} & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{V}^\top = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

$$\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3, \boldsymbol{u}_4], \Sigma = \mathrm{diag}(3, \sqrt{5}, 2, 0), \boldsymbol{V}^\top = \begin{bmatrix} \boldsymbol{v}_1^\top \\ \boldsymbol{v}_2^\top \\ \boldsymbol{v}_3^\top \\ \boldsymbol{v}_4^\top \\ \boldsymbol{v}_5^\top \end{bmatrix}$$

$$\boldsymbol{A} = \sum_{i=1}^{4} \sigma_{ii} \boldsymbol{u}_i \boldsymbol{v}_i^{\top}$$

$$\boldsymbol{A} = 3 \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \end{bmatrix} + 3 \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \end{bmatrix} + 2 \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

- Compact SVD

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{U} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \Sigma = \begin{bmatrix} 3 & 0 & 0 \\ 0 & \sqrt{5} & 0 \\ 0 & 0 & 2 \end{bmatrix}, \boldsymbol{V}^\top = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{U} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{5} & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{V}^\top = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

$\Sigma$ (the singular values) is unique, while $\boldsymbol{U}$ (left singular vectors) and $\boldsymbol{V}$ (right singular vectors) are not unique.

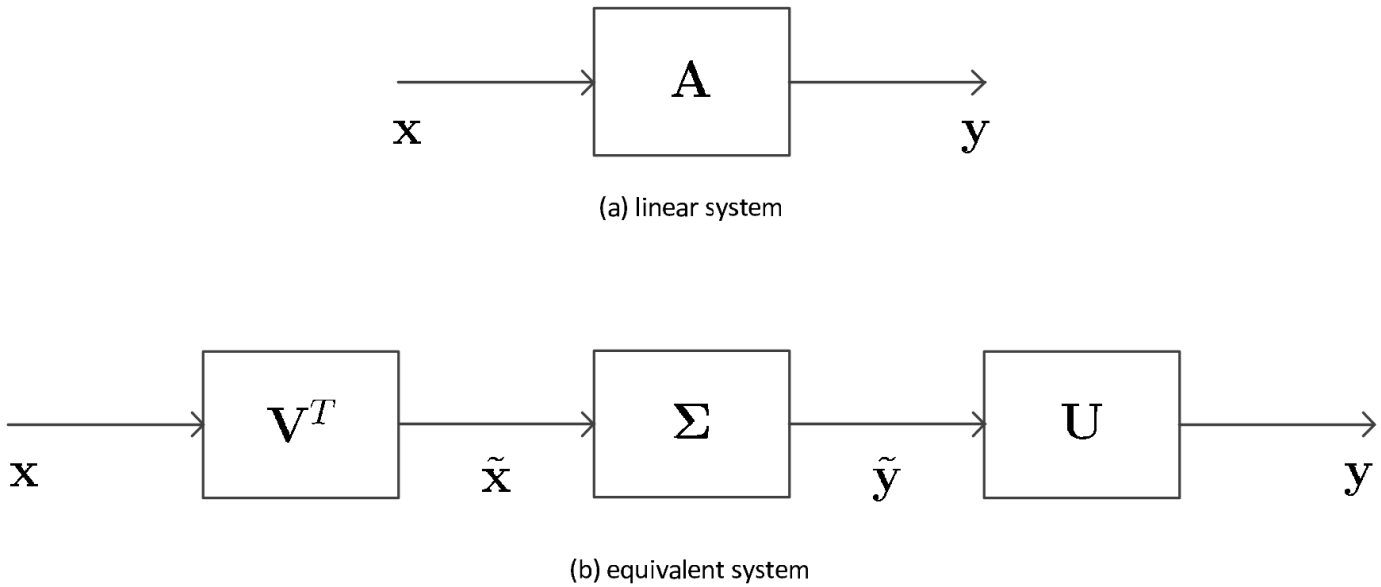



Figure: Matrix-vector product from the perspective of SVD [3].

[3] http://www.ee.cuhk.edu.hk/~wkma/

# Power iteration

## Purpose

Power iteration is an algorithm to compute:

- The dominant eigenvalue $\lambda_1$ (largest in magnitude)
- The corresponding eigenvector $\boldsymbol{v}_1$

## Key Assumption

- $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$
- Where $\lambda_i$ are the eigenvalues of $\boldsymbol{A}$
- Convergence rate depends on $|\lambda_2/\lambda_1|$

# Algorithm description

**Algorithm 2** Power Iteration

1: Choose initial vector $\boldsymbol{x}_0$ randomly, with $\|\boldsymbol{x}_0\| = 1$
2: **for** $k = 1, 2, \ldots$ until convergence **do**
3: $\quad \boldsymbol{q}_k = \boldsymbol{A}\boldsymbol{x}_{k-1}$
4: $\quad \boldsymbol{x}_k = \boldsymbol{q}_k / \|\boldsymbol{q}_k\|$
5: $\quad \lambda_k = \boldsymbol{x}_k^\top \boldsymbol{A}\boldsymbol{x}_k$ ▷ Rayleigh quotient
6: **end for**

# Power iteration

```python
#Demonstrate code for power interation
import torch

# Test matrix
X = torch.tensor([[4., 1.], [1., 3.]])

# Power iteration
v = torch.randn(2)
for _ in range(20):
    v = X @ v
    v = v / torch.norm(v)

eigenvalue = (v @ X @ v) / (v @ v)
print(f"λ={eigenvalue:.4f}, v={v}")
U,S,V = torch.svd(X)
print(S)
```

```
λ=4.6180, v=tensor([-0.8507, -0.5257])
tensor([4.6180, 2.3820])
```

Figure: Demonstrate code for power iteration.

## Matrix-vector dot product

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

$$\boldsymbol{x} = [-\sqrt{0.2}, 0, 2, 0, -\sqrt{0.8}]^\top$$

$$\boldsymbol{y} = \underbrace{-2}_{\boldsymbol{v}_1^\top \boldsymbol{x}} \times \underbrace{0.3}_{\sigma_1} \times \underbrace{[0, -1, 0, 0]^\top}_{\boldsymbol{u}_1} + \underbrace{1}_{\boldsymbol{v}_2^\top \boldsymbol{x}} \times \underbrace{0.2}_{\sigma_2} \times \underbrace{[-1, 0, 0, 0]^\top}_{\boldsymbol{u}_2}$$

## Matrix-vector dot product

$$\boldsymbol{y} = \boldsymbol{B}\boldsymbol{x}$$

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{x} = [-\sqrt{0.2}, 0, 2, 0, -\sqrt{0.8}]^\top$$

$$\boldsymbol{U} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{5} & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{V}^\top = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

$$\boldsymbol{y} = -2 * 3 * [0, -1, 0, 0]^\top + 1 * \sqrt{5} * [-1, 0, 0, 0]^\top$$

## Matrix-vector dot product

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}$$

$$\boldsymbol{C} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ -\sqrt{0.2} & 0 & 0 & 0 & -\sqrt{0.8} \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix}$$

$$\boldsymbol{y} = -2 * 3 * [0, -1, 0, 0]^\top + 1 * 2 * [-1, 0, 0, 0]^\top$$

# Vector norms

$$\|\boldsymbol{x}\|_1 = \sum_i |x_i|$$

$$\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}$$

$$\|\boldsymbol{x}\|_p = \left[\sum_i |x_i|^p\right]^{1/p}$$

$$\|\boldsymbol{x}\|_\infty = \max_i |x_i|$$

# Matrix norms

$$\|\boldsymbol{A}\|_1 = \max_j \sum_i |A_{ij}|$$

$$\|\boldsymbol{A}\|_2 = \sqrt{\max \operatorname{eig}(\boldsymbol{A}^\top \boldsymbol{A})}$$

$$\|\boldsymbol{A}\|_p = \left( \max_{\|\boldsymbol{x}\|_p = 1} \|\boldsymbol{A}\boldsymbol{x}\|_p \right)$$

$$\|\boldsymbol{A}\|_\infty = \max_i \sum_j |A_{ij}|$$

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} |A_{ij}|^2} = \sqrt{\operatorname{trace}(\boldsymbol{A}^\top \boldsymbol{A})}$$

# Inequalities

Assuming $\boldsymbol{A}$ is an $m \times n$, and $d = \text{rank}(\boldsymbol{A})$

| | $\|\boldsymbol{A}\|_{\max}$ | $\|\boldsymbol{A}\|_1$ | $\|\boldsymbol{A}\|_\infty$ | $\|\boldsymbol{A}\|_2$ | $\|\boldsymbol{A}\|_F$ | $\|\boldsymbol{A}\|_{\text{KF}}$ |
|---|---|---|---|---|---|---|
| $\|\boldsymbol{A}\|_{\max}$ | $1$ | $1$ | $1$ | $1$ | $1$ | $1$ |
| $\|\boldsymbol{A}\|_1$ | $m$ | $1$ | $m$ | $\sqrt{m}$ | $\sqrt{m}$ | $\sqrt{m}$ |
| $\|\boldsymbol{A}\|_\infty$ | $n$ | $n$ | $1$ | $\sqrt{n}$ | $\sqrt{n}$ | $\sqrt{n}$ |
| $\|\boldsymbol{A}\|_2$ | $\sqrt{mn}$ | $\sqrt{n}$ | $\sqrt{m}$ | $1$ | $1$ | $1$ |
| $\|\boldsymbol{A}\|_F$ | $\sqrt{mn}$ | $\sqrt{n}$ | $\sqrt{m}$ | $\sqrt{d}$ | $1$ | $1$ |
| $\|\boldsymbol{A}\|_{\text{KF}}$ | $\sqrt{mnd}$ | $\sqrt{nd}$ | $\sqrt{md}$ | $d$ | $\sqrt{d}$ | $1$ |

which are to be read as, e.g.

$$\|\boldsymbol{A}\|_2 \leq \sqrt{m} \cdot \|\boldsymbol{A}\|_\infty$$

# Theorem: Singular Value Bounds of a Gaussian Random Matrix

## Theorem

*Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ have i.i.d. standard Gaussian entries, i.e., $\boldsymbol{A}_{i,j} \overset{iid}{\sim} \mathcal{N}(0,1)$. For every $m \geq n$, we have the following inequality:*

$$\sqrt{m} - \sqrt{n} \leq \mathbb{E}[\sigma_{\min}(\boldsymbol{A})] \leq \mathbb{E}[\sigma_{\max}(\boldsymbol{A})] \leq \sqrt{m} + \sqrt{n},$$

*where $\sigma_{\min}(\boldsymbol{A})$ and $\sigma_{\max}(\boldsymbol{A})$ denote the minimal and maximal singular values, respectively.*

# Lemma: Upper Bound of Weight Matrix Norm (Xavier Initialization)

## Lemma

*Let* $\boldsymbol{A} \in \mathbb{R}^{n_{in} \times n_{out}}$ *have i.i.d. standard Gaussian entries, i.e.,*

$$A_{i,j} \overset{iid}{\sim} \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right).$$

*We have the following inequality for its maximum singular value:*

$$\mathbb{E}[\sigma_{\max}(\boldsymbol{A})] \leq 2.$$

$$A_{i,j} \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \left(\frac{1}{\sqrt{n_{in}} + \sqrt{n_{out}}}\right)^2\right)$$

# Expected norm of addition of two unit random vectors

$$\boldsymbol{x}_1 = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}, \boldsymbol{x} \text{ is a random high-dimensional vector.}$$

$$\boldsymbol{y}_1 = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2}, \boldsymbol{y} \text{ is a random high-dimensional vector.}$$

$$\boldsymbol{z}_1 = \boldsymbol{x}_1 + \boldsymbol{y}_1$$

$$\mathbb{E}(\|\boldsymbol{z}\|_2) = \sqrt{2}$$

# Expected norm of addition of two unit random vectors

```python
#Demonstrate code for power interation
import torch
# Set parameters
dim = 1024        # dimension of vectors
n_trials = 1000   # number of trials
# Store norms of results
result_norms = torch.zeros(n_trials)
# Perform trials
for i in range(n_trials):
    # Create two random vectors
    v1 = torch.randn(dim)
    v2 = torch.randn(dim)

    # L2 normalize both vectors
    v1_normalized = v1 / torch.norm(v1)
    v2_normalized = v2 / torch.norm(v2)

    # Add normalized vectors and compute norm
    result = v1_normalized + v2_normalized
    result_norms[i] = torch.norm(result)
# Calculate statistics
mean_norm = torch.mean(result_norms)
var_norm = torch.var(result_norms)

print(f"Statistics for {n_trials} trials with {dim}-dimensional vectors:")
print(f"Mean norm: {mean_norm:.6f}")
print(f"Variance of norm: {var_norm:.6f}")
print(f"Standard deviation of norm: {torch.sqrt(var_norm):.6f}")
```

```
Statistics for 1000 trials with 1024-dimensional vectors:
Mean norm: 1.413116
Variance of norm: 0.000480
Standard deviation of norm: 0.021916
```

Figure: Demonstrate code for expected norm of addition of two unit vectors.

# Expected norm under orthogonal transformation

- Given: $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ where $\boldsymbol{A}$ is orthogonal
- Properties of orthogonal matrix $\boldsymbol{A}$:

$$\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}$$

- Consider the norm of $\boldsymbol{y}$:

$$\begin{aligned} \|\boldsymbol{y}\|^2 &= \boldsymbol{y}^\top \boldsymbol{y} = (\boldsymbol{A}\boldsymbol{x})^\top (\boldsymbol{A}\boldsymbol{x}) \\ &= \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^\top \boldsymbol{x} = \|\boldsymbol{x}\|^2 \end{aligned}$$

- Taking expectations:

$$\mathbb{E}[\|\boldsymbol{y}\|^2] = \mathbb{E}[\|\boldsymbol{x}\|^2]$$

- Therefore:

$$\mathbb{E}[\|\boldsymbol{y}\|] = \mathbb{E}[\|\boldsymbol{x}\|]$$

# Expected Norm with Orthogonal Transformation

```python
import torch

# Parameters
n = 100  # dimension
trials = 1000
norms = torch.zeros(trials)
# Create orthogonal matrix using QR decomposition
A = torch.randn(n, n)
Q, _ = torch.linalg.qr(A)  # Q is orthogonal
for i in range(trials):
    # Random input vector
    x = torch.randn(n)
    x = x / torch.norm(x)

    # Compute norm of y = Ax
    norms[i] = torch.norm(Q @ x)

print(f"Expected norm: {torch.mean(norms):.4f} ± {torch.std(norms):.4f}")
print(f"Input vector norm (mean): {torch.mean(torch.tensor(norms[i])):.4f}")
```

```
Expected norm: 1.0000 ± 0.0000
Input vector norm (mean): 1.0000
```

Figure: Expected Norm with L2 Normalized Columns.

# Expected norm with L2 normalized columns

- Given: $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ with L2 normalized columns
- Properties of $\boldsymbol{A}$:

$$\|\boldsymbol{a}_i\|_2 = 1 \text{ for all columns } i$$

$$\boldsymbol{A} = [\boldsymbol{a}_1 \ \boldsymbol{a}_2 \ \cdots \ \boldsymbol{a}_n]$$

- Consider the norm squared of $\boldsymbol{y}$:

$$\begin{aligned} \|\boldsymbol{y}\|^2 = \boldsymbol{y}^\top \boldsymbol{y} = (\boldsymbol{A}\boldsymbol{x})^\top (\boldsymbol{A}\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} \\ = \boldsymbol{x}^\top \boldsymbol{M}\boldsymbol{x} \text{ where } \boldsymbol{M} = \boldsymbol{A}^\top \boldsymbol{A} \end{aligned}$$

- Properties of $\boldsymbol{M} = \boldsymbol{A}^\top \boldsymbol{A}$:
  - $M_{ii} = 1$ (due to L2 normalization)
  - $M_{ij} = \boldsymbol{a}_i^\top \boldsymbol{a}_j$ (inner products of columns)
- Therefore:

$$\mathbb{E}[\|\boldsymbol{y}\|^2] = \mathbb{E}[\boldsymbol{x}^\top \boldsymbol{M}\boldsymbol{x}]$$

$$= \mathbb{E}[\|\boldsymbol{x}\|^2] + \sum_{i \neq j} M_{ij} \mathbb{E}[x_i x_j]$$

# Expected Norm with L2 Normalized Columns

```python
import torch

# Parameters
n = 100  # dimension of square matrix
trials = 1000
norms = torch.zeros(trials)
# Create matrix A with L2 normalized columns
A = torch.randn(n, n)
A = A / torch.norm(A, dim=0, keepdim=True)  # normalize columns
for i in range(trials):
    # Random input vector
    x = torch.randn(n)
    x = x / torch.norm(x)

    # Compute norm of y = Ax
    norms[i] = torch.norm(A @ x)

print(f"Expected norm: {torch.mean(norms):.4f} ± {torch.std(norms):.4f}")
U,S,V = torch.svd(A)
print('Expected norm: ', torch.sqrt(torch.pow(S, 2).mean()))
```

```
Expected norm: 0.9944 ± 0.0689
Expected norm:  tensor(1.)
```

Figure: Expected Norm with L2 Normalized Columns.

# Expected norm with L2 normalized rows

- Given: $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ where:
- $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with L2 normalized rows
- Properties of $\boldsymbol{A}$: $\|\boldsymbol{a}_i^\top\|_2 = 1$ for all rows $i$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^\top \\ \boldsymbol{a}_2^\top \\ \vdots \\ \boldsymbol{a}_m^\top \end{bmatrix}$$

- Consider $\|\boldsymbol{y}\|^2$: $\|\boldsymbol{y}\|^2 = \sum_{i=1}^m y_i^2 = \sum_{i=1}^m (\boldsymbol{a}_i^\top \boldsymbol{x})^2 = \sum_{i=1}^m \boldsymbol{x}^\top (\boldsymbol{a}_i \boldsymbol{a}_i^\top) \boldsymbol{x}$
- $\mathbb{E}[\|\boldsymbol{y}\|^2] = \sum_{i=1}^m \mathbb{E}[\boldsymbol{x}^\top (\boldsymbol{a}_i \boldsymbol{a}_i^\top) \boldsymbol{x}] = \sum_{i=1}^m \left(\mathbb{E}[\|\boldsymbol{x}\|^2] \cdot \frac{1}{n} + \text{correlation terms}\right)$

  Note:
- Each row contributes $\|\boldsymbol{x}\|^2/n$ in expectation if $\boldsymbol{x}$ is isotropic
- Total contribution depends on row correlations

# Expected Norm with L2 Normalized Rows

```python
#Demonstrate code to Expected Norm with L2 Normalized Rows (Rectangle)
import torch

# Parameters
m, n = 100, 100
trials = 1000
norms = torch.zeros(trials)
A = torch.randn(m, n)
A = A / torch.norm(A, dim=1, keepdim=True)
for i in range(trials):
    # Create and normalize A and x
    x = torch.randn(n)
    x = x / torch.norm(x)

    # Compute norm of y = Ax
    norms[i] = torch.norm(A @ x)

print(f"Expected norm: {torch.mean(norms):.4f} ± {torch.std(norms):.4f}")
U,S,V = torch.svd(A)
print('Expected norm: ', torch.sqrt(torch.pow(S, 2).mean()))
```

```
Expected norm: 0.9981 ± 0.0699
Expected norm:  tensor(1.0000)
```

Figure: Expected Norm with L2 Normalized Rows (Rectangle).

# Expected norm with L2 normalized row (underdetermined)

- Given: $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ where:
- $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m < n$ and L2 normalized rows
- Properties of $\boldsymbol{A}$: $\|\boldsymbol{a}_i^\top\|_2 = 1$ for all rows $i$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^\top \\ \boldsymbol{a}_2^\top \\ \vdots \\ \boldsymbol{a}_m^\top \end{bmatrix}$$

- Consider $\|\boldsymbol{y}\|^2$: $\|\boldsymbol{y}\|^2 = \sum_{i=1}^m y_i^2 = \sum_{i=1}^m (\boldsymbol{a}_i^\top \boldsymbol{x})^2 = \sum_{i=1}^m \boldsymbol{x}^\top (\boldsymbol{a}_i \boldsymbol{a}_i^\top) \boldsymbol{x}$
- $\mathbb{E}[\|\boldsymbol{y}\|^2] = \sum_{i=1}^m \mathbb{E}[\boldsymbol{x}^\top (\boldsymbol{a}_i \boldsymbol{a}_i^\top) \boldsymbol{x}] = \sum_{i=1}^m \left(\mathbb{E}[\|\boldsymbol{x}\|^2] \cdot \frac{1}{n} + \text{correlation terms}\right)$

Note:

- Each row contributes $\|\boldsymbol{x}\|^2/n$ in expectation if $\boldsymbol{x}$ is isotropic
- Total contribution depends on row correlations

# Expected Norm with L2 Normalized Rows

```python
#Demonstrate code to Expected Norm with L2 Normalized Rows (Underdetermined)
import torch

# Parameters
m, n = 20, 100
trials = 1000
norms = torch.zeros(trials)
A = torch.randn(m, n)
A = A / torch.norm(A, dim=1, keepdim=True)
for i in range(trials):
    # Create and normalize A and x
    x = torch.randn(n)
    x = x / torch.norm(x)

    # Compute norm of y = Ax
    norms[i] = torch.norm(A @ x)

print(f"Expected norm: {torch.mean(norms):.4f} ± {torch.std(norms):.4f}")
U,S,V = torch.svd(A)
print('Expected norm: ', torch.sqrt(torch.pow(S, 2).mean()))
```

```
Expected norm: 0.4389 ± 0.0697
Expected norm:  tensor(1.)
```

Figure: Expected Norm with L2 Normalized Rows (Underdetermined).

# Expected norm with L2 normalized row (overdetermined)

- Given: $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m > n$ and L2 normalized rows
  Thus, we have

$$\|\boldsymbol{a}_i^\top\|_2 = 1 \text{ for all rows } i, \boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^\top \\ \boldsymbol{a}_2^\top \\ \vdots \\ \boldsymbol{a}_m^\top \end{bmatrix}$$

- Consider $\|\boldsymbol{y}\|_2^2$: $\|\boldsymbol{y}\|_2^2 = \sum_{i=1}^m y_i^2 = \sum_{i=1}^m (\boldsymbol{a}_i^\top \boldsymbol{x})^2 = \sum_{i=1}^m \boldsymbol{x}^\top (\boldsymbol{a}_i \boldsymbol{a}_i^\top) \boldsymbol{x}$
- Taking expectations:

$$\mathbb{E}[\|\boldsymbol{y}\|^2] = \sum_{i=1}^m \mathbb{E}[\boldsymbol{x}^\top (\boldsymbol{a}_i \boldsymbol{a}_i^\top) \boldsymbol{x}] = \sum_{i=1}^m \left( \mathbb{E}[\|\boldsymbol{x}\|^2] \cdot \frac{1}{n} + \text{correlation terms} \right)$$

- Key observations:
  - $\mathbb{E}[\|\boldsymbol{y}\|^2]$ typically larger than $\mathbb{E}[\|\boldsymbol{x}\|^2]$ due to redundancy

# Expected Norm with L2 Normalized Rows

```python
#Demonstrate code to Expected Norm with L2 Normalized Rows
import torch

# Parameters
m, n = 100, 20
trials = 1000
norms = torch.zeros(trials)
A = torch.randn(m, n)
A = A / torch.norm(A, dim=1, keepdim=True)
for i in range(trials):
    # Create and normalize A and x
    x = torch.randn(n)
    x = x / torch.norm(x)

    # Compute norm of y = Ax
    norms[i] = torch.norm(A @ x)

print(f"Expected norm: {torch.mean(norms):.4f} ± {torch.std(norms):.4f}")
U,S,V = torch.svd(A)
print('Expected norm: ', torch.sqrt(torch.pow(S, 2).mean()))
```

```
Expected norm: 2.2343 ± 0.1432
Expected norm:  tensor(2.2361)
```

Figure: Expected Norm with L2 Normalized Rows (Overdetermined).

# Eigenvalue decomposition & SVD: a close relationship

- **EVD vs SVD**:

$$\text{EVD: } \boldsymbol{A} = \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{-1}, \text{ where } \boldsymbol{A} \text{ is } n \times n$$

$$\text{SVD: } \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top}, \text{ where } \boldsymbol{A} \text{ is } m \times n$$

- **Key Connections**:
  - For symmetric matrices $\boldsymbol{A}$:

$$\boldsymbol{A}\boldsymbol{A}^{\top} = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top})(\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^{\top}) = \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^{\top}$$

  - Singular values ($\sigma_i$) relate to eigenvalues ($\lambda_i$):

$$\sigma_i = \sqrt{\lambda_i(\boldsymbol{A}\boldsymbol{A}^{\top})}$$

- **Important Implications**:
  - Left singular vectors ($\boldsymbol{U}$) are eigenvectors of $\boldsymbol{A}\boldsymbol{A}^{\top}$
  - Right singular vectors ($\boldsymbol{V}$) are eigenvectors of $\boldsymbol{A}^{\top}\boldsymbol{A}$

# Part 2: Matrix Calculus
# [7, 9, 16, 10, 11]

# Single-variable calculus[4]

**Basic Properties and Formulas**

If $f(x)$ and $g(x)$ are differentiable functions (the derivative exists), $c$ and $n$ are any real numbers,

1. $\frac{d}{dx}\left(c\right) = 0$
2. $\left(c\,f(x)\right)' = c\,f'(x)$
3. $\frac{d}{dx}\left(x^n\right) = n\,x^{n-1}$ – **Power Rule**
4. $\left(f(x) \pm g(x)\right)' = f'(x) \pm g'(x)$
5. $\left(f(x)\,g(x)\right)' = f'(x)\,g(x) + f(x)\,g'(x)$ – **Product Rule**
6. $\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)\,g(x) - f(x)\,g'(x)}{\left(g(x)\right)^2}$ – **Quotient Rule**
7. $\frac{d}{dx}\left(f\left(g(x)\right)\right) = f'\left(g(x)\right)\,g'(x)$ – **Chain Rule**

[4] https://tutorial.math.lamar.edu

**Common Derivatives**

$$\frac{d}{dx}\Big(x\Big) = 1$$

$$\frac{d}{dx}\Big(\sin(x)\Big) = \cos(x)$$

$$\frac{d}{dx}\Big(\cos(x)\Big) = -\sin(x)$$

$$\frac{d}{dx}\Big(\tan(x)\Big) = \sec^2(x)$$

$$\frac{d}{dx}\Big(\sec(x)\Big) = \sec(x)\tan(x)$$

$$\frac{d}{dx}\Big(\csc(x)\Big) = -\csc(x)\cot(x)$$

$$\frac{d}{dx}\Big(\cot(x)\Big) = -\csc^2(x)$$

$$\frac{d}{dx}\Big(\sin^{-1}(x)\Big) = \frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\Big(\cos^{-1}(x)\Big) = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\Big(\tan^{-1}(x)\Big) = \frac{1}{1+x^2}$$

$$\frac{d}{dx}\Big(a^x\Big) = a^x \ln(a)$$

$$\frac{d}{dx}\Big(\mathbf{e}^x\Big) = \mathbf{e}^x$$

$$\frac{d}{dx}\Big(\ln(x)\Big) = \frac{1}{x}, \quad x > 0$$

$$\frac{d}{dx}\Big(\ln|x|\Big) = \frac{1}{x}, \quad x \neq 0$$

$$\frac{d}{dx}\Big(\log_a(x)\Big) = \frac{1}{x\ln(a)}, \quad x > 0$$

**Chain Rule Variants**

The chain rule applied to some specific functions.

1. $\frac{d}{dx}\left(\left[f(x)\right]^n\right) = n\left[f(x)\right]^{n-1} f'(x)$
2. $\frac{d}{dx}\left(\mathbf{e}^{f(x)}\right) = f'(x)\,\mathbf{e}^{f(x)}$
3. $\frac{d}{dx}\left(\ln\left[f(x)\right]\right) = \frac{f'(x)}{f(x)}$
4. $\frac{d}{dx}\left(\sin\left[f(x)\right]\right) = f'(x)\cos\left[f(x)\right]$
5. $\frac{d}{dx}\left(\cos\left[f(x)\right]\right) = -f'(x)\sin\left[f(x)\right]$
6. $\frac{d}{dx}\left(\tan\left[f(x)\right]\right) = f'(x)\sec^2\left[f(x)\right]$
7. $\frac{d}{dx}\left(\sec\left[f(x)\right]\right) = f'(x)\sec\left[f(x)\right]\tan\left[f(x)\right]$
8. $\frac{d}{dx}\left(\tan^{-1}\left[f(x)\right]\right) = \frac{f'(x)}{1+\left[f(x)\right]^2}$

**Higher Order Derivatives**

The $2^{nd}$ Derivative is denoted as $f''(x) = f^{(2)}(x) = \frac{d^2 f}{dx^2}$ and is defined as $f''(x) = \left(f'(x)\right)'$, *i.e.* the derivative of the first derivative, $f'(x)$.

The $n^{th}$ Derivative is denoted as $f^{(n)}(x) = \frac{d^n f}{dx^n}$ and is defined as $f^{(n)}(x) = \left(f^{(n-1)}(x)\right)'$, *i.e.* the derivative of the $(n-1)^{st}$ derivative, $f^{(n-1)}(x)$.

# Matrix calculus: numerator vs. denominator layout [5]

- Consider the chain rule expression:

$$\frac{d\boldsymbol{z}}{d\boldsymbol{x}} = \frac{d\boldsymbol{z}}{d\boldsymbol{y}}\frac{d\boldsymbol{y}}{d\boldsymbol{x}} \stackrel{?}{=} \frac{d\boldsymbol{y}}{d\boldsymbol{x}}\frac{d\boldsymbol{z}}{d\boldsymbol{y}}$$

- Two main conventions:
  1. Numerator Layout
     - ★ Elements of derivative arranged by numerator
     - ★ $[\frac{d\boldsymbol{z}}{d\boldsymbol{x}}]_{ij} = \frac{\partial z_i}{\partial x_j}$
     - ★ Chain rule: left-to-right multiplication
  2. Denominator Layout
     - ★ Elements of derivative arranged by denominator
     - ★ $[\frac{d\boldsymbol{z}}{d\boldsymbol{x}}]_{ij} = \frac{\partial z_j}{\partial x_i}$
     - ★ Chain rule: right-to-left multiplication
- Choice affects matrix multiplication order in chain rule!

[5] Matrix calculus wiki

# Numerator layout in matrix calculus

- Case 1: Scalar by Vector ($y$ scalar, $\boldsymbol{x}$ vector)

$$\frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

- Case 2: Vector by Scalar ($\boldsymbol{y}$ vector, $x$ scalar)

$$\frac{\partial \boldsymbol{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$$

# Numerator layout in matrix calculus

- Case 3: Vector by Vector ($\boldsymbol{y}$ vector, $\boldsymbol{x}$ vector)

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

- Case 4: Scalar by Matrix ($y$ scalar, $\boldsymbol{X}$ matrix)

$$\frac{\partial y}{\partial \boldsymbol{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{p1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{p2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1q}} & \frac{\partial y}{\partial x_{2q}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix}$$

# Denominator layout in matrix calculus

- Case 1: Scalar by Vector ($y$ scalar, $\boldsymbol{x}$ vector)

$$\frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix}$$

- Case 2: Vector by Scalar ($\boldsymbol{y}$ vector, $x$ scalar)

$$\frac{\partial \boldsymbol{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \cdots & \frac{\partial y_m}{\partial x} \end{bmatrix}$$

# Denominator layout in matrix calculus

- Case 3: Vector by Vector ($\boldsymbol{y}$ vector, $\boldsymbol{x}$ vector)

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

- Case 4: Scalar by Matrix ($y$ scalar, $\boldsymbol{X}$ matrix)

$$\frac{\partial y}{\partial \boldsymbol{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} & \cdots & \frac{\partial y}{\partial x_{1q}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{2q}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{p1}} & \frac{\partial y}{\partial x_{p2}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix}$$

# Derivatives and Gradients

- For a scalar-valued function $f(\cdot) : \boldsymbol{x} \to f(\boldsymbol{x})$
- The derivative is a $1 \times m$ row vector:

$$\frac{\partial f}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_m} \end{bmatrix}$$

- The gradient $\nabla_{\boldsymbol{x}} f$ is the transposed column vector:

$$\nabla_{\boldsymbol{x}} f = \left( \frac{\partial f}{\partial \boldsymbol{x}} \right)^{\top} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{bmatrix}$$

# Jacobian matrix

- For a vector-valued function $\boldsymbol{f}(\cdot) : \boldsymbol{x} \rightarrow \boldsymbol{f}(\boldsymbol{x})$
- The derivative, known as the Jacobian, is an $n \times m$ matrix
- The $(i, j)$-th element is the scalar derivative of the $i$-th output w.r.t the $j$-th input:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

# Scalar by matrix derivatives

- For a scalar-valued function $f(\cdot) : \underset{m\times n}{\boldsymbol{X}} \to \underset{1\times 1}{f(\boldsymbol{X})}$
- The derivative is an $m \times n$ matrix:

$$\frac{\partial f}{\partial \boldsymbol{X}} = \begin{bmatrix} \frac{\partial f}{\partial X_{1,1}} & \cdots & \frac{\partial f}{\partial X_{m,1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial X_{1,n}} & \cdots & \frac{\partial f}{\partial X_{m,n}} \end{bmatrix}$$

- The gradient has dimensions matching the input matrix: $m \times n$

# Matrix-by-scalar derivatives

- For a matrix-valued function $\underset{1\times 1}{x} \to \underset{p\times q}{\boldsymbol{F}(x)}$
- The derivative is a $p \times q$ matrix:

$$\frac{\partial \boldsymbol{F}}{\partial x} = \begin{bmatrix} \frac{\partial F_{1,1}}{\partial x} & \cdots & \frac{\partial F_{1,q}}{\partial x} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_{p,1}}{\partial x} & \cdots & \frac{\partial F_{p,q}}{\partial x} \end{bmatrix}$$

- Each element is the scalar derivative of the corresponding matrix element

# Vector-by-matrix derivatives

- For a vector-valued function $\text{vec}(\cdot) : \underset{m\times n}{\boldsymbol{X}} \to \underset{p\times 1}{\boldsymbol{f}(\boldsymbol{X})}$
- The derivative is a 3rd-order tensor with dimensions $p \times m \times n$:

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{X}} = \begin{bmatrix} \frac{\partial \boldsymbol{f}}{\partial X_{1,1}} & \cdots & \frac{\partial \boldsymbol{f}}{\partial X_{m,1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{f}}{\partial X_{1,n}} & \cdots & \frac{\partial \boldsymbol{f}}{\partial X_{m,n}} \end{bmatrix}$$

- Each element $\frac{\partial \boldsymbol{f}}{\partial X_{i,j}}$ is a $p$-dimensional vector
- This represents a stack of $p$ matrices, each of size $m \times n$

# Matrix-by-vector derivatives

- For a matrix-valued function $\text{vec}(\cdot) : \underset{m\times 1}{\boldsymbol{x}} \to \underset{p\times q}{\boldsymbol{F}(\boldsymbol{x})}$
- The derivative is a 3rd-order tensor with dimensions $p \times q \times m$:

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} = \left[\frac{\partial \boldsymbol{F}}{\partial x_1} \quad \frac{\partial \boldsymbol{F}}{\partial x_2} \quad \cdots \quad \frac{\partial \boldsymbol{F}}{\partial x_m}\right]$$

- Each element $\frac{\partial \boldsymbol{F}}{\partial x_i}$ is a $p \times q$ matrix
- The tensor can be viewed as $m$ matrices of size $p \times q$

# Tensor derivatives

- For a vector-valued function $\boldsymbol{f}(\cdot) : \underset{m\times n}{\boldsymbol{X}} \to \underset{p\times q}{\boldsymbol{F}(\boldsymbol{X})}$
- The derivative is a 4th-order tensor with dimensions $p \times q \times m \times n$

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{X}} = \begin{bmatrix} \frac{\partial \boldsymbol{F}}{\partial X_{1,1}} & \cdots & \frac{\partial \boldsymbol{F}}{\partial X_{m,1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{F}}{\partial X_{1,n}} & \cdots & \frac{\partial \boldsymbol{F}}{\partial X_{m,n}} \end{bmatrix}$$

- This represents $p \times q$ matrices of size $m \times n$, one for each output dimension

# Gradient and Hessian [6]

The oracle is

$$f = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x}$$

The gradient (first-order derivative) is

$$\nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = (\boldsymbol{A} + \boldsymbol{A}^\top)\boldsymbol{x} + \boldsymbol{b}$$

The Hessian (second-order derivative) is

$$\frac{\partial^2 f}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\top} = \boldsymbol{A} + \boldsymbol{A}^\top$$

[6] Matrix calculus calculator

# Gradient and Hessian

The oracle is

$$\begin{aligned} f &= (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})^\top (\boldsymbol{C}\boldsymbol{x} + \boldsymbol{d}) \\ &= \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{C}\boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{d} + \boldsymbol{b}^\top \boldsymbol{C}\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{d} \end{aligned}$$

The gradient (first-order derivative) is

$$\nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = (\boldsymbol{A}^\top \boldsymbol{C} + \boldsymbol{C}^\top \boldsymbol{A})\boldsymbol{x} + \boldsymbol{A}^\top \boldsymbol{d} + \boldsymbol{C}^\top \boldsymbol{b}$$

The Hessian (second-order derivative) is

$$\frac{\partial^2 f}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\top} = \boldsymbol{A}^\top \boldsymbol{C} + \boldsymbol{C}^\top \boldsymbol{A}$$

# Gradient example: L2 norm

- Problem: Find $\nabla_{\boldsymbol{x}} f$ where $f(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{a}\|_2$
- Note: $\|\boldsymbol{x} - \boldsymbol{a}\|_2 = \sqrt{(\boldsymbol{x} - \boldsymbol{a})^\top (\boldsymbol{x} - \boldsymbol{a})}$ is scalar
- Using chain rule:

$$\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial \sqrt{(\boldsymbol{x} - \boldsymbol{a})^\top (\boldsymbol{x} - \boldsymbol{a})}}{\partial (\boldsymbol{x} - \boldsymbol{a})^\top (\boldsymbol{x} - \boldsymbol{a})} \cdot \frac{\partial (\boldsymbol{x} - \boldsymbol{a})^\top (\boldsymbol{x} - \boldsymbol{a})}{\partial \boldsymbol{x}}$$

- Where:
  - First term $= \frac{1}{2\sqrt{(\boldsymbol{x}-\boldsymbol{a})^\top (\boldsymbol{x}-\boldsymbol{a})}}$
  - Second term $= 2(\boldsymbol{x} - \boldsymbol{a})^\top$
- Final result:

$$\frac{\partial f}{\partial \boldsymbol{x}} = \frac{(\boldsymbol{x} - \boldsymbol{a})^\top}{\sqrt{(\boldsymbol{x} - \boldsymbol{a})^\top (\boldsymbol{x} - \boldsymbol{a})}}$$

# Derivative of normalized vector

- Problem: Find $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ where $\boldsymbol{y} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$
- Using quotient rule and chain rule:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\partial}{\partial \boldsymbol{x}} \left( \frac{\boldsymbol{x}}{\sqrt{\boldsymbol{x}^\top \boldsymbol{x}}} \right)$$

$$= \frac{\boldsymbol{I}\sqrt{\boldsymbol{x}^\top \boldsymbol{x}} - \boldsymbol{x} \frac{\partial}{\partial \boldsymbol{x}} \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}}{(\sqrt{\boldsymbol{x}^\top \boldsymbol{x}})^2}$$

- Note that $\frac{\partial}{\partial \boldsymbol{x}} \sqrt{\boldsymbol{x}^\top \boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$
- Final result:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{1}{\|\boldsymbol{x}\|_2} \left( \boldsymbol{I} - \frac{\boldsymbol{x}\boldsymbol{x}^\top}{\boldsymbol{x}^\top \boldsymbol{x}} \right)$$

- This is a projection matrix scaled by $\frac{1}{\|\boldsymbol{x}\|_2}$

# Norm derivative with respect to $\boldsymbol{x}$

## Matrix Calculus

MatrixCalculus provides matrix calculus for everyone. It is an online tool that computes vector and matrix derivatives (matrix calculus).

derivative of `(x-a)/norm2(x-a)` w.r.t. x

$$\frac{\partial}{\partial x}((x-a)/\|x-a\|_2) = 1/t_1 \cdot \mathbb{I} - 1/t_1^3 \cdot t_0 \cdot t_0^\top$$

where

$$t_0 = x - a$$
$$t_1 = \|t_0\|_2$$

where

a is a vector

x is a vector

Export functions as

Python Latex

Common subexpressions

ON

Figure: Expected Norm with L2 Normalized Columns.

# Derivative of least-square

- Problem: Find $\frac{\partial f}{\partial \boldsymbol{x}}$ where $f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2$
- Expanding the squared norm:

$$f(\boldsymbol{x}) = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$$
$$= \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{y} - \boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{y}^\top \boldsymbol{y}$$

- Taking the derivative:

$$\frac{\partial f}{\partial \boldsymbol{x}} = 2\boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A} - 2\boldsymbol{y}^\top \boldsymbol{A}$$

- Final result can also be written as:

$$\frac{\partial f}{\partial \boldsymbol{x}} = 2(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{A}$$

- This is the gradient used in linear least squares optimization

# Least-square derivative with respect to $y$

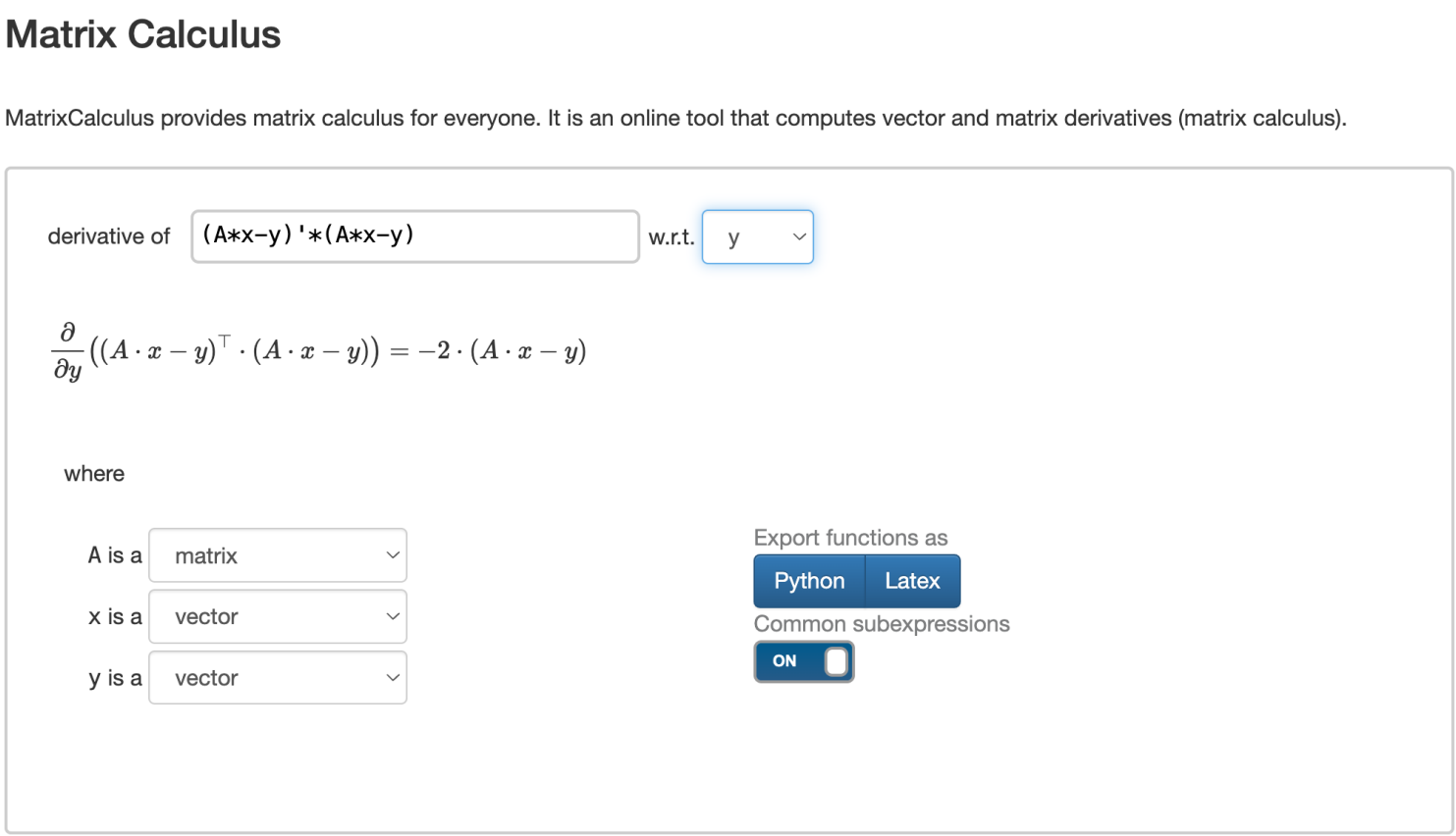



Figure: Expected Norm with L2 Normalized Columns.

# Least-square derivative with respect to $x$

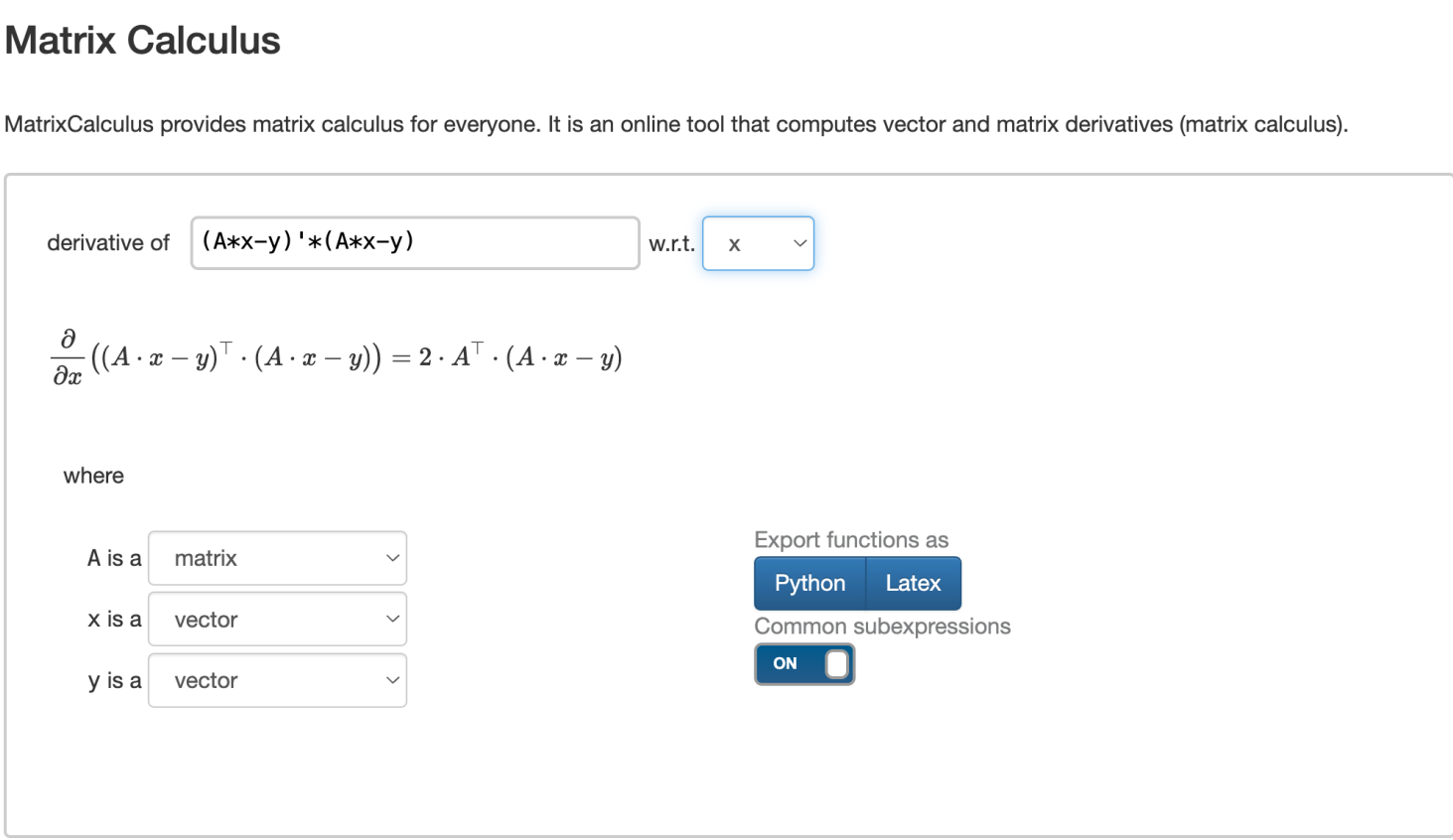



Figure: Expected Norm with L2 Normalized Columns.

# Least-square derivative with respect to $A$

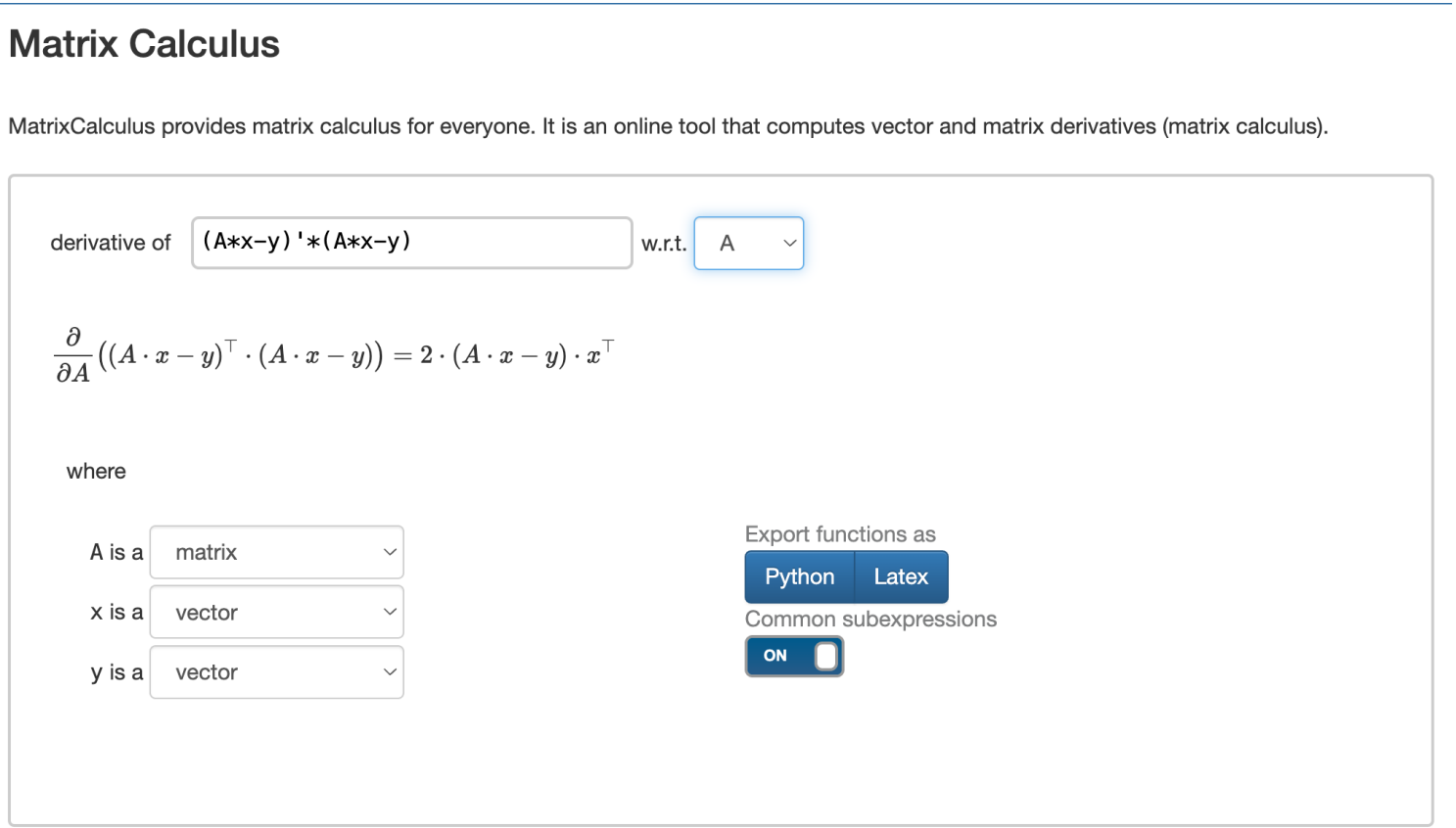



Figure: Expected Norm with L2 Normalized Columns.

# Extended examples of matrix calculus

- Basic vector operations:
  - $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{x}, \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = 2\boldsymbol{x}$
  - $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2, \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$
  - $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2, \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = 2\boldsymbol{x}$
- Vector transformations:
  - $\boldsymbol{y} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}, \quad \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right) = \frac{1}{\|\boldsymbol{x}\|_2}(\boldsymbol{I} - \frac{\boldsymbol{x}\boldsymbol{x}^\top}{\boldsymbol{x}^\top \boldsymbol{x}})$
  - $f(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{a}\|_2, \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = \frac{(\boldsymbol{x}-\boldsymbol{a})}{\|\boldsymbol{x}-\boldsymbol{a}\|_2}$
- Matrix-vector operations:
  - $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}, \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = (\boldsymbol{A} + \boldsymbol{A}^\top)\boldsymbol{x}$
  - $f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|^2, \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = 2\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$
  - $f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x}\|_2, \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = \frac{\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x}}{\|\boldsymbol{A}\boldsymbol{x}\|_2}$
- Common objective functions:
  - $f(\boldsymbol{x}) = \log(1 + e^{\boldsymbol{x}^\top \boldsymbol{a}}), \quad \nabla_{\boldsymbol{x}} f = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^\top = \frac{e^{\boldsymbol{x}^\top \boldsymbol{a}}}{1+e^{\boldsymbol{x}^\top \boldsymbol{a}}}\boldsymbol{a}$
  - $f(\boldsymbol{x}) = -\sum_i y_i \log(x_i), \quad \frac{\partial f}{\partial \boldsymbol{x}} = -[\frac{y_1}{x_1}, \ldots, \frac{y_n}{x_n}]^\top$

# Vec operator

- For matrix product $\boldsymbol{AB}$:
  $\boldsymbol{A} \in \mathcal{R}^{m \times n}, \boldsymbol{B} \in \mathcal{R}^{n \times k}$

$$\text{vec}(\boldsymbol{AB}) = (\boldsymbol{I}_k \otimes \boldsymbol{A})\,\text{vec}(\boldsymbol{B}) = (\boldsymbol{B}^\top \otimes \boldsymbol{I}_m)\,\text{vec}(\boldsymbol{A})$$

- Properties:
  - $\otimes$ denotes Kronecker product
  - $\boldsymbol{I}_k$ is $k \times k$ identity matrix
  - $\boldsymbol{I}_m$ is $m \times m$ identity matrix

# Vec operator derivatives: two matrices

- Let $\boldsymbol{P} = \boldsymbol{A}\boldsymbol{B}$ where:
  - $\boldsymbol{A} \in \mathcal{R}^{m \times n}$
  - $\boldsymbol{B} \in \mathcal{R}^{n \times k}$

- Derivatives:

$$\frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{A})} = \boldsymbol{B}^{\top} \otimes \boldsymbol{I}_m$$

$$\frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{B})} = \boldsymbol{I}_k \otimes \boldsymbol{A}$$

# Vec operator: triple matrices

- Let $\boldsymbol{P} = \boldsymbol{ABC}$ where:
  - $\boldsymbol{A} \in \mathcal{R}^{m \times n}$
  - $\boldsymbol{B} \in \mathcal{R}^{n \times k}$
  - $\boldsymbol{C} \in \mathcal{R}^{k \times l}$

$$\begin{aligned}
\text{vec}(\boldsymbol{ABC}) &= (\boldsymbol{C}^\top \boldsymbol{B}^\top \otimes \boldsymbol{I}_m)\,\text{vec}(\boldsymbol{A}) \\
&= (\boldsymbol{C}^\top \otimes \boldsymbol{A})\,\text{vec}(\boldsymbol{B}) \\
&= (\boldsymbol{I}_l \otimes \boldsymbol{AB})\,\text{vec}(\boldsymbol{C})
\end{aligned}$$

# Vec operator derivatives: triple matrices

- Let $\boldsymbol{P} = \boldsymbol{ABC}$ where:
  - $\boldsymbol{A} \in \mathcal{R}^{m \times n}$
  - $\boldsymbol{B} \in \mathcal{R}^{n \times k}$
  - $\boldsymbol{C} \in \mathcal{R}^{k \times l}$

- Derivatives:

$$\frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{A})} = \boldsymbol{C}^\top \boldsymbol{B}^\top \otimes \boldsymbol{I}_m$$

$$\frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{B})} = \boldsymbol{C}^\top \otimes \boldsymbol{A}$$

$$\frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{C})} = \boldsymbol{I}_l \otimes \boldsymbol{AB}$$

# First-order matrix derivatives: basic cases

- Basic trace derivatives:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{X}) = \boldsymbol{I}$$

- With constant matrix $\boldsymbol{A}$:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{X}\boldsymbol{A}) = \boldsymbol{A}^\top$$

- With transpose variations:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{X}^\top \boldsymbol{A}) = \boldsymbol{A}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{A}\boldsymbol{X}^\top) = \boldsymbol{A}$$

# First-order matrix derivatives: advanced cases

- With two constant matrices $\boldsymbol{A}, \boldsymbol{B}$:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}) = \boldsymbol{A}^\top \boldsymbol{B}^\top$$

- Transpose variations:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{A}\boldsymbol{X}^\top \boldsymbol{B}) = \boldsymbol{B}\boldsymbol{A}$$

- Kronecker product:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{A} \otimes \boldsymbol{X}) = \text{trace}(\boldsymbol{A})\boldsymbol{I}$$

# $AXB$ derivative respect to $X$

## Matrix Calculus

MatrixCalculus provides matrix calculus for everyone. It is an online tool that computes vector and matrix derivatives (matrix calculus).

derivative of `tr(A*X*B)` w.r.t. X

$$\frac{\partial}{\partial X}(\mathrm{tr}(A \cdot X \cdot B)) = (B \cdot A)^\top$$

where

A is a matrix

B is a matrix

X is a matrix

Export functions as

Python Latex

Common subexpressions

ON

Figure: trace($AXB$) derivative respect to $X$.

# Second-order matrix derivatives: basic cases

- Quadratic term:

$$\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}^2) = 2\boldsymbol{X}^\top$$

- With single constant matrix $\boldsymbol{B}$:

$$\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}^2\boldsymbol{B}) = (\boldsymbol{X}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{X})^\top$$

- Equivalent forms:

$$\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}^\top\boldsymbol{X}) = \frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}\boldsymbol{X}^\top) = 2\boldsymbol{X}$$

- With quadratic term:

$$\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}^\top\boldsymbol{B}\boldsymbol{X}) = \boldsymbol{B}\boldsymbol{X} + \boldsymbol{B}^\top\boldsymbol{X}$$

# Second-order matrix derivatives: advanced cases

- Various forms with $\boldsymbol{X}\boldsymbol{X}^\top$:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{B}\boldsymbol{X}\boldsymbol{X}^\top) = \boldsymbol{B}\boldsymbol{X} + \boldsymbol{B}^\top \boldsymbol{X}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{B}) = \boldsymbol{B}\boldsymbol{X} + \boldsymbol{B}^\top \boldsymbol{X}$$

- Forms with $\boldsymbol{X}\boldsymbol{B}\boldsymbol{X}^\top$:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{X}\boldsymbol{B}\boldsymbol{X}^\top) = \boldsymbol{X}\boldsymbol{B}^\top + \boldsymbol{X}\boldsymbol{B}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{B}\boldsymbol{X}^\top \boldsymbol{X}) = \boldsymbol{X}\boldsymbol{B}^\top + \boldsymbol{X}\boldsymbol{B}$$

- Double matrix product:

$$\frac{\partial}{\partial \boldsymbol{X}} \text{trace}(\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}\boldsymbol{X}) = \boldsymbol{A}^\top \boldsymbol{X}^\top \boldsymbol{B}^\top + \boldsymbol{B}^\top \boldsymbol{X}^\top \boldsymbol{A}^\top$$

# trace($\boldsymbol{BXX}^\top$) derivative respect to $\boldsymbol{X}$

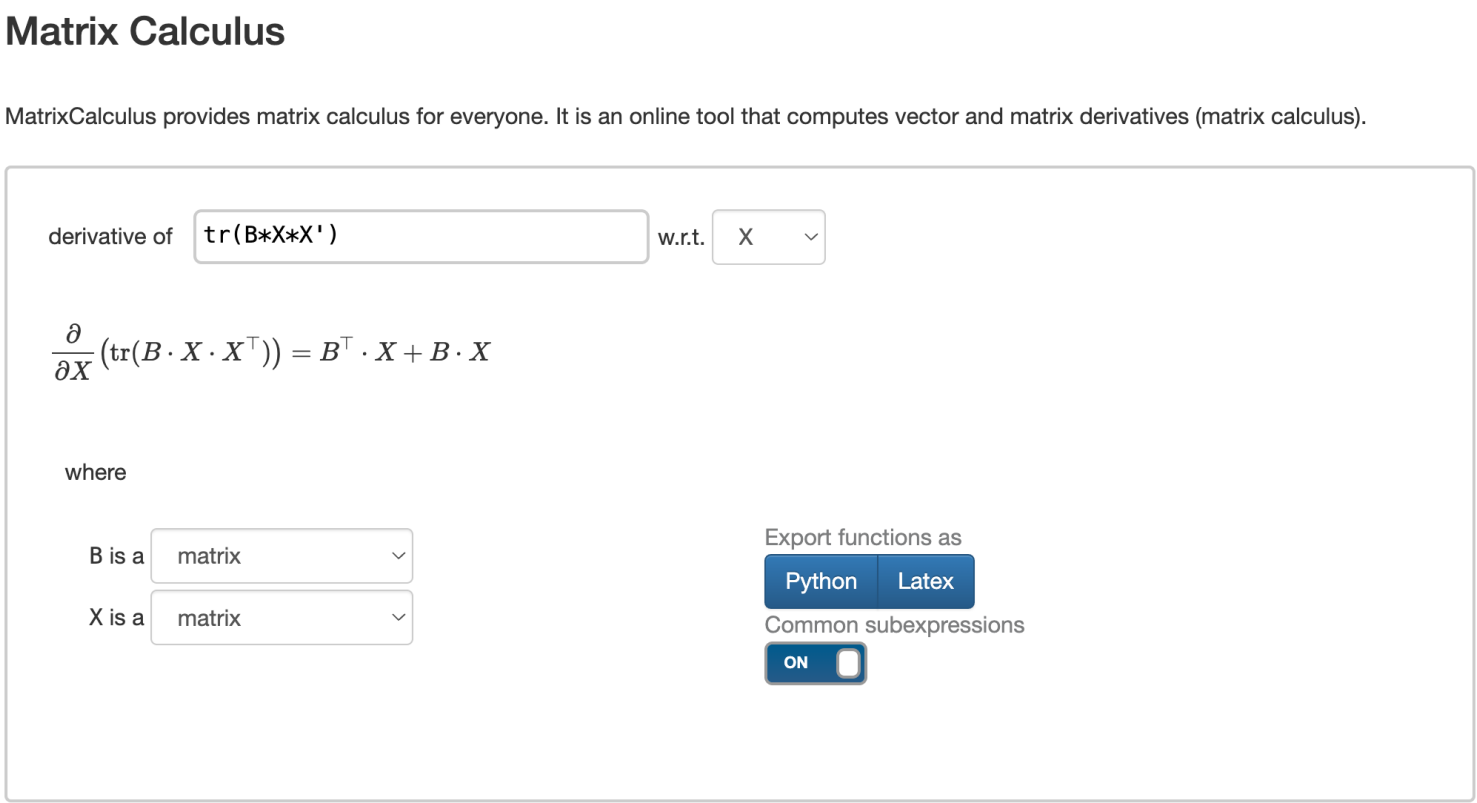



Figure: trace($\boldsymbol{BXX}^\top$) derivative respect to $\boldsymbol{X}$.

# Higher-order matrix derivatives: k-th power cases

- General k-th power rule:

$$\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}^k) = k(\boldsymbol{X}^{k-1})^\top$$

- With constant matrix $\boldsymbol{A}$:

$$\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{A}\boldsymbol{X}^k) = \sum_{r=0}^{k-1} (\boldsymbol{X}^r \boldsymbol{A} \boldsymbol{X}^{k-r-1})^\top$$

- Special cases:
  - $k = 2$: $\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}^2) = 2\boldsymbol{X}^\top$
  - $k = 3$: $\frac{\partial}{\partial \boldsymbol{X}} \operatorname{trace}(\boldsymbol{X}^3) = 3(\boldsymbol{X}^2)^\top$

# Higher-order matrix derivatives: complex cases

- Complex trace derivative with multiple matrices:

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{X}}\text{trace}[\boldsymbol{B}^\top \boldsymbol{X}^\top \boldsymbol{C}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{C}\boldsymbol{X}\boldsymbol{B}] = \\ \boldsymbol{C}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{C}\boldsymbol{X}\boldsymbol{B}\boldsymbol{B}^\top \\ + \boldsymbol{C}^\top \boldsymbol{X}\boldsymbol{B}\boldsymbol{B}^\top \boldsymbol{X}^\top \boldsymbol{C}^\top \boldsymbol{X} \\ + \boldsymbol{C}\boldsymbol{X}\boldsymbol{B}\boldsymbol{B}^\top \boldsymbol{X}^\top \boldsymbol{C}\boldsymbol{X} \\ + \boldsymbol{C}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{C}^\top \boldsymbol{X}\boldsymbol{B}\boldsymbol{B}^\top\end{aligned}$$

# Theorem: differential product rule

## Theorem (Differential Product Rule)

*Let* $\boldsymbol{A}, \boldsymbol{B}$ *be two matrices. Then, we have the differential product rule for* $\boldsymbol{AB}$*:*

$$d(\boldsymbol{AB}) = (d\boldsymbol{A})\boldsymbol{B} + \boldsymbol{A}(d\boldsymbol{B})$$

- By the differential of matrix $\boldsymbol{A}$, we think of it as a small (unconstrained) change in matrix $\boldsymbol{A}$
- Later, constraints may be placed on the allowed perturbations

# Differential operator: scalar case

- For a scalar function $f(x)$, the differential is:

$$df = f(x + dx) - f(x)$$

- This represents an infinitesimal change in $f$ due to an infinitesimal change in $x$
- Key properties:
  - $d(f + g) = df + dg$
  - $d(fg) = (df)g + f(dg)$ (product rule)
  - $d(f/g) = \frac{(df)g - f(dg)}{g^2}$ (quotient rule)

# Differential operator: vector case

- For a vector $\boldsymbol{x}$, the differential is:

$$d\boldsymbol{x} = \begin{bmatrix} dx_1 \\ dx_2 \\ \vdots \\ dx_n \end{bmatrix}$$

- Properties extend component-wise:
  - $d(\boldsymbol{x} + \boldsymbol{y}) = d\boldsymbol{x} + d\boldsymbol{y}$
  - $d(\boldsymbol{x}^\top \boldsymbol{y}) = (d\boldsymbol{x})^\top \boldsymbol{y} + \boldsymbol{x}^\top (d\boldsymbol{y})$
  - $d\|\boldsymbol{x}\| = \frac{\boldsymbol{x}^\top d\boldsymbol{x}}{\|\boldsymbol{x}\|}$
- The differential represents a small displacement in vector space

# Differential operator: matrix case

- For a matrix $\boldsymbol{A}$, the differential is:

$$d\boldsymbol{A} = \begin{bmatrix} dA_{11} & dA_{12} & \cdots & dA_{1n} \\ dA_{21} & dA_{22} & \cdots & dA_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ dA_{m1} & dA_{m2} & \cdots & dA_{mn} \end{bmatrix}$$

- Key matrix differential rules:
  - $d(\boldsymbol{AB}) = (d\boldsymbol{A})\boldsymbol{B} + \boldsymbol{A}(d\boldsymbol{B})$
  - $d(\boldsymbol{A}^\top) = (d\boldsymbol{A})^\top$
  - $d(\boldsymbol{A}^{-1}) = -\boldsymbol{A}^{-1}(d\boldsymbol{A})\boldsymbol{A}^{-1}$

# Differential operator: tensor case

- For a tensor $\boldsymbol{A}$, the differential applies element-wise:

$$dA_{ijkl\ldots} = A_{ijkl\ldots}(x + dx) - A_{ijkl\ldots}(x)$$

- Properties:
  - Linearity: $d(\alpha\boldsymbol{A} + \beta\boldsymbol{B}) = \alpha d\boldsymbol{A} + \beta d\boldsymbol{B}$
  - Product Rule extends to tensor products
  - Maintains tensor order and structure
- Applications:
  - Stress/strain analysis in continuum mechanics
  - Multi-dimensional data analysis
  - Higher-order derivatives in deep learning

# Differential operator: basic rules

- Constant rule:

$$\partial \boldsymbol{A} = 0$$

- Linear operations:

$$\partial(\alpha \boldsymbol{X}) = \alpha \partial \boldsymbol{X}$$

$$\partial(\boldsymbol{X} + \boldsymbol{Y}) = \partial \boldsymbol{X} + \partial \boldsymbol{Y}$$

- Product rules:

$$\partial(\boldsymbol{X}\boldsymbol{Y}) = (\partial \boldsymbol{X})\boldsymbol{Y} + \boldsymbol{X}(\partial \boldsymbol{Y})$$

$$\partial(\boldsymbol{X} \circ \boldsymbol{Y}) = (\partial \boldsymbol{X}) \circ \boldsymbol{Y} + \boldsymbol{X} \circ (\partial \boldsymbol{Y})$$

$$\partial(\boldsymbol{X} \otimes \boldsymbol{Y}) = (\partial \boldsymbol{X}) \otimes \boldsymbol{Y} + \boldsymbol{X} \otimes (\partial \boldsymbol{Y})$$

- Transpose and Hermitian:

$$\partial \boldsymbol{X}^{\top} = (\partial \boldsymbol{X})^{\top}$$

$$\partial \boldsymbol{X}^{H} = (\partial \boldsymbol{X})^{H}$$

# Differential operator: advanced rules

- Trace operation:
$$\partial(\text{trace}(\boldsymbol{X})) = \text{trace}(\partial \boldsymbol{X})$$
- Matrix inverse:
$$\partial(\boldsymbol{X}^{-1}) = -\boldsymbol{X}^{-1}(\partial \boldsymbol{X})\boldsymbol{X}^{-1}$$
- Determinant (two forms):
$$\partial(\det(\boldsymbol{X})) = \text{trace}(\text{adj}(\boldsymbol{X})\partial \boldsymbol{X})$$
$$\partial(\det(\boldsymbol{X})) = \det(\boldsymbol{X})\,\text{trace}(\boldsymbol{X}^{-1}\partial \boldsymbol{X})$$
- Log-determinant:
$$\partial(\ln(\det(\boldsymbol{X}))) = \text{trace}(\boldsymbol{X}^{-1}\partial \boldsymbol{X})$$
- Note: These rules are essential in:
  - Matrix optimization
  - Neural network backpropagation
  - Statistical parameter estimation

# Matrix trace differentives

$$
\begin{aligned}
d\,\mathrm{trace}(\boldsymbol{AXBX}^\top \boldsymbol{C}) &= d\,\mathrm{trace}(\boldsymbol{CAXBX}^\top) = \mathrm{trace}(d(\boldsymbol{CAXBX}^\top)) \\
&= \mathrm{trace}(\boldsymbol{CAX}d(\boldsymbol{BX}^\top) + d(\boldsymbol{CAX})\boldsymbol{BX}^\top) \\
&= \mathrm{trace}(\boldsymbol{CAX}d(\boldsymbol{BX}^\top)) + \mathrm{trace}(d(\boldsymbol{CAX})\boldsymbol{BX}^\top) \\
&= \mathrm{trace}(\boldsymbol{CAXB}d(\boldsymbol{X}^\top)) + \mathrm{trace}(\boldsymbol{CA}(d\boldsymbol{X})\boldsymbol{BX}^\top) \\
&= \mathrm{trace}(\boldsymbol{CAXB}(d\boldsymbol{X})^\top) + \mathrm{trace}(\boldsymbol{CA}(d\boldsymbol{X})\boldsymbol{BX}^\top) \\
&= \mathrm{trace}((\boldsymbol{CAXB}(d\boldsymbol{X})^\top)^\top) + \mathrm{trace}(\boldsymbol{CA}(d\boldsymbol{X})\boldsymbol{BX}^\top) \\
&= \mathrm{trace}((d\boldsymbol{X})\boldsymbol{B}^\top\boldsymbol{X}^\top\boldsymbol{A}^\top\boldsymbol{C}^\top) + \mathrm{trace}(\boldsymbol{CA}(d\boldsymbol{X})\boldsymbol{BX}^\top) \\
&= \mathrm{trace}(\boldsymbol{B}^\top\boldsymbol{X}^\top\boldsymbol{A}^\top\boldsymbol{C}^\top(d\boldsymbol{X})) + \mathrm{trace}(\boldsymbol{BX}^\top\boldsymbol{CA}(d\boldsymbol{X})) \\
&= \mathrm{trace}((\boldsymbol{B}^\top\boldsymbol{X}^\top\boldsymbol{A}^\top\boldsymbol{C}^\top + \boldsymbol{BX}^\top\boldsymbol{CA})d\boldsymbol{X}) \\
&= \mathrm{trace}((\boldsymbol{CAXB} + \boldsymbol{A}^\top\boldsymbol{C}^\top\boldsymbol{XB}^\top)^\top d\boldsymbol{X})
\end{aligned}
$$

# Lipschitz continuity

## Definition

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz continuous if there exists $L \geq 0$ such that:

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. The smallest such $L$ is called the Lipschitz constant.

## Linear transformations

For a linear transformation $\boldsymbol{A} : \mathbb{R}^n \to \mathbb{R}^m$:

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}\boldsymbol{y}\| = \|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|$$

# Matrix norms and Lipschitz constants

## Key Relationship

For a matrix $\boldsymbol{A}$, its Lipschitz constant equals its operator norm:

$$L = \|\boldsymbol{A}\| = \sup_{\boldsymbol{x} \neq 0} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

## Matrix norms

- Spectral norm (2-norm): $\|\boldsymbol{A}\|_2 = \sqrt{\lambda_{\max}(\boldsymbol{A}^\top \boldsymbol{A})}$
- Frobenius norm: $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$
- Maximum absolute row sum: $\|\boldsymbol{A}\|_\infty = \max_i \sum_j |a_{ij}|$
- Maximum absolute column sum: $\|\boldsymbol{A}\|_1 = \max_j \sum_i |a_{ij}|$

# Properties

## Basic Properties

For matrices $\boldsymbol{A}$ and $\boldsymbol{B}$:

- $\|\boldsymbol{AB}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|$
- $\|c\boldsymbol{A}\| = |c|\|\boldsymbol{A}\|$ for scalar $c$
- $\|\boldsymbol{A}\| \geq 0$ with equality iff $\boldsymbol{A} = 0$
- $\|\boldsymbol{A}\| \geq \rho(\boldsymbol{A})$ (spectral radius)

## Relation to singular values

For the spectral norm:

$$\|\boldsymbol{A}\|_2 = \sigma_1(\boldsymbol{A})$$

where $\sigma_1$ is the largest singular value of $\boldsymbol{A}$

# Applications in optimization

## Gradient Descent

For function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}$:

- Gradient: $\nabla f(\boldsymbol{x}) = \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$
- Lipschitz constant: $L = \|\boldsymbol{A}^\top \boldsymbol{A}\|_2 = \sigma_1^2(\boldsymbol{A})$
- Step size bound: $\alpha < \frac{2}{L}$ for convergence

## Convergence Analysis

Rate of convergence depends on condition number:

$$\kappa(\boldsymbol{A}) = \frac{\sigma_1(\boldsymbol{A})}{\sigma_n(\boldsymbol{A})} = \frac{L}{\mu}$$

where $\mu$ is strong convexity parameter

# Practical computation

## Computing Lipschitz Constants

For matrix $\boldsymbol{A}$:

1. SVD method: $L = \sigma_1(\boldsymbol{A})$
2. Power iteration for $\boldsymbol{A}^\top \boldsymbol{A}$
3. Direct norm computation for simple cases

## Example

For $2 \times 2$ matrix:

$$\boldsymbol{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Spectral norm (Lipschitz constant):

$$L = \sqrt{\lambda_{\max}(\boldsymbol{A}^\top \boldsymbol{A})} = \sqrt{\frac{\alpha + \sqrt{\alpha^2 - 4\beta}}{2}}$$

where $\alpha = a^2 + b^2 + c^2 + d^2$ and $\beta = (ad - bc)^2$

# Part 3: Transformer [25, 17, 18, 12, 19, 20, 29, 27, 2]

# Transformer

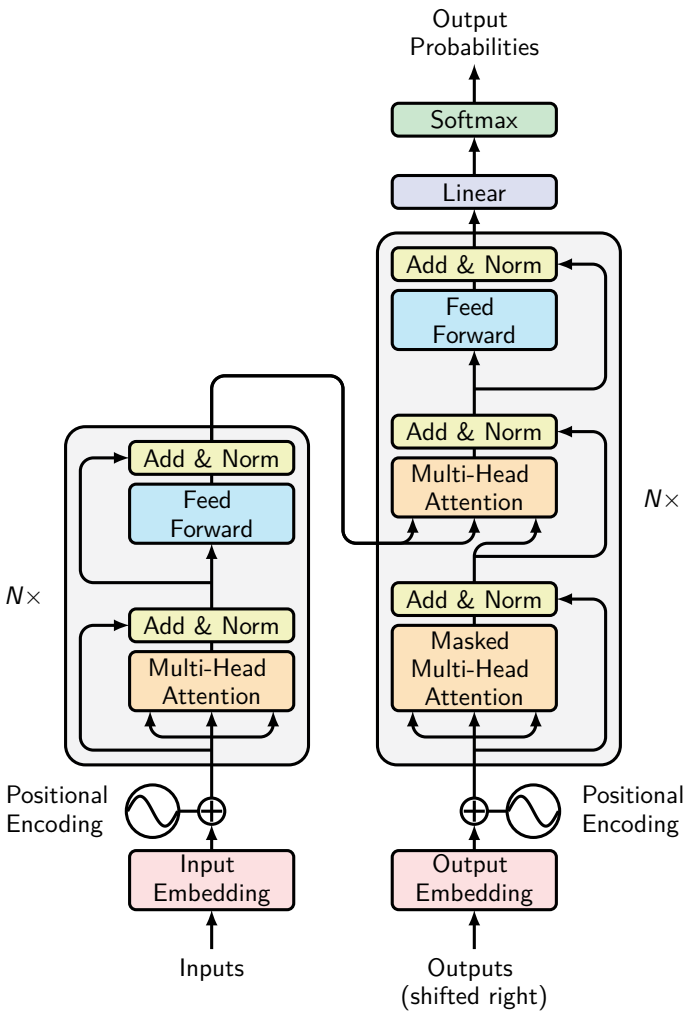



Figure: Original Transformer

# Pure-decoder

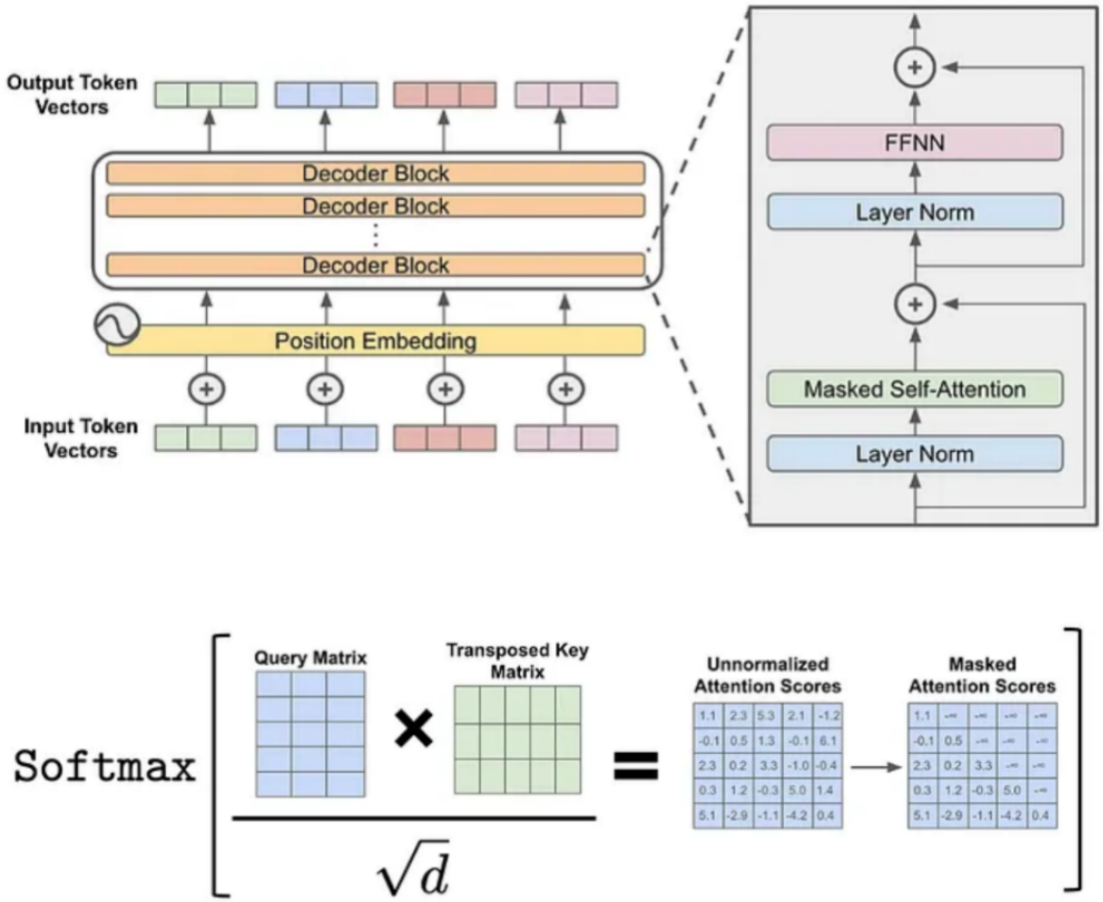



Figure: GPT.

# Encoder

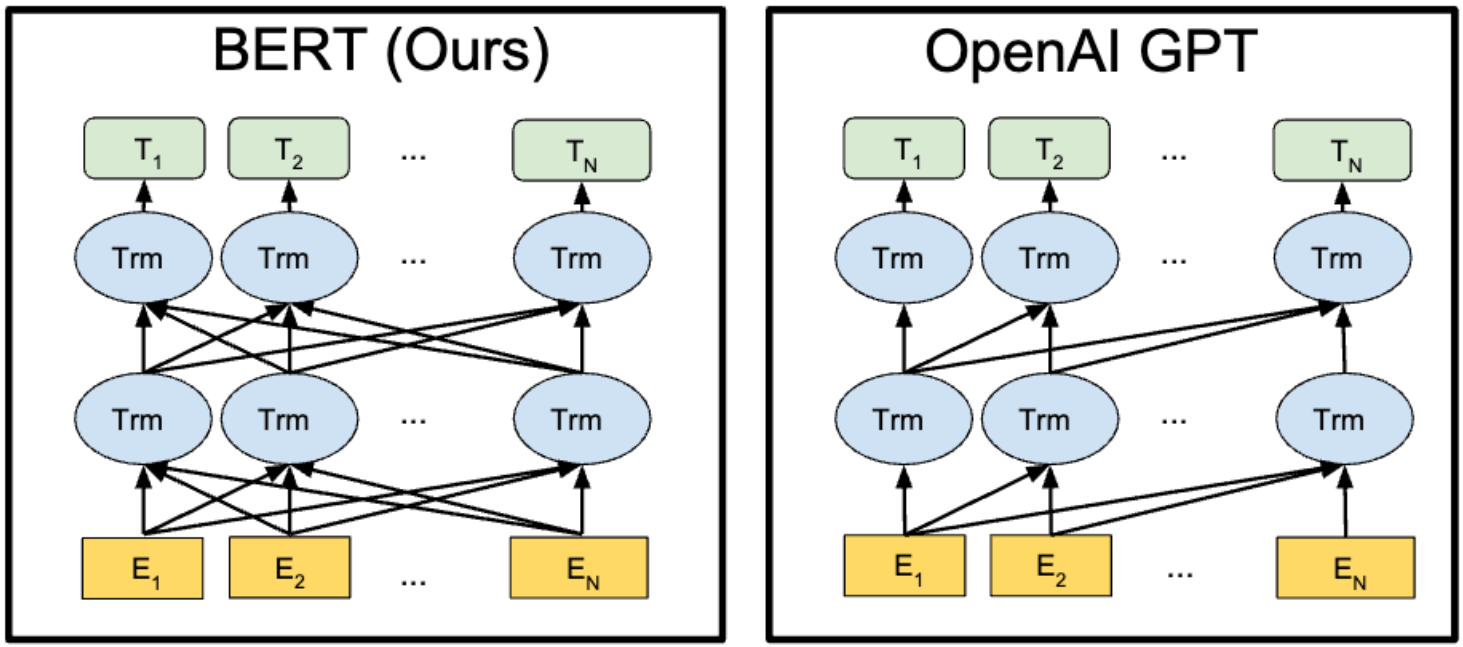



Figure: BERT vs GPT.

# Pure-encoder

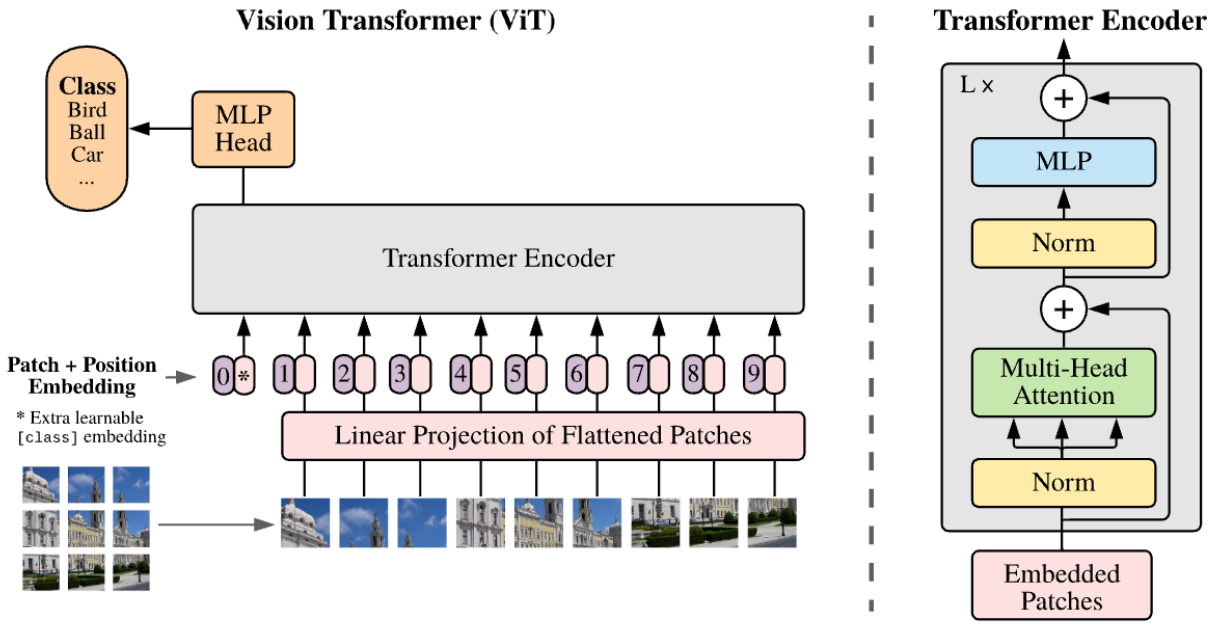



Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Figure: Vision Transformer.

# Pure-encoder

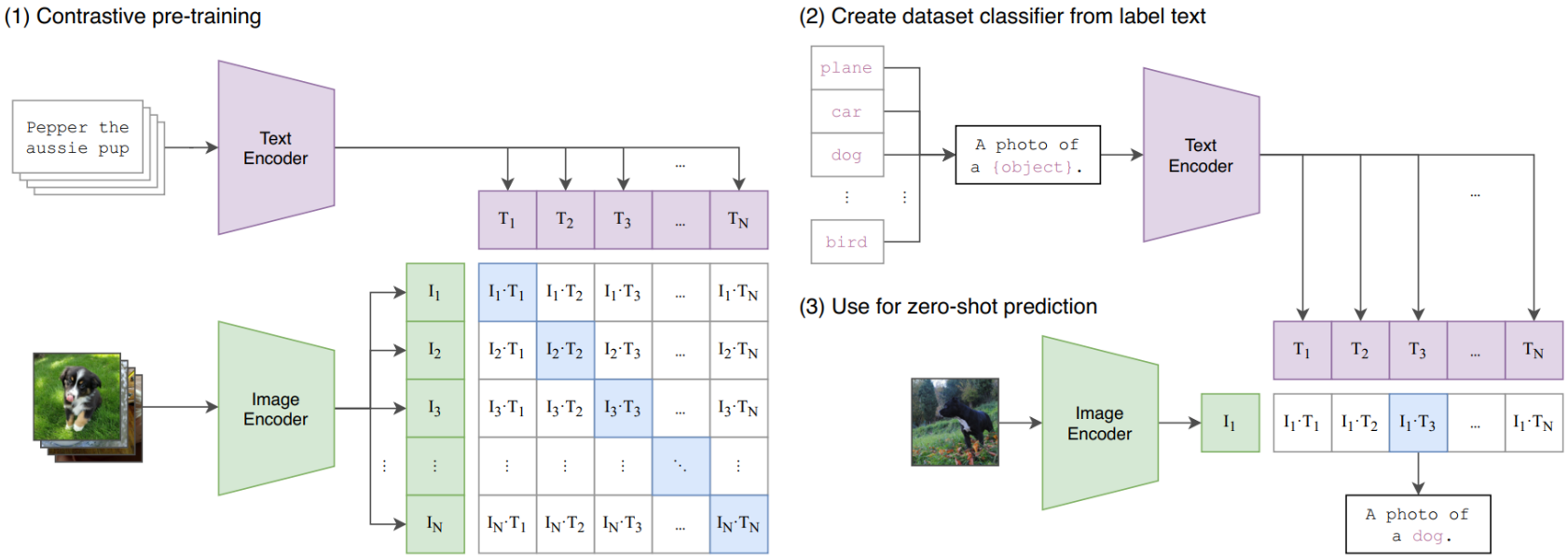



Figure: CLIP.

# Encoder-decoder

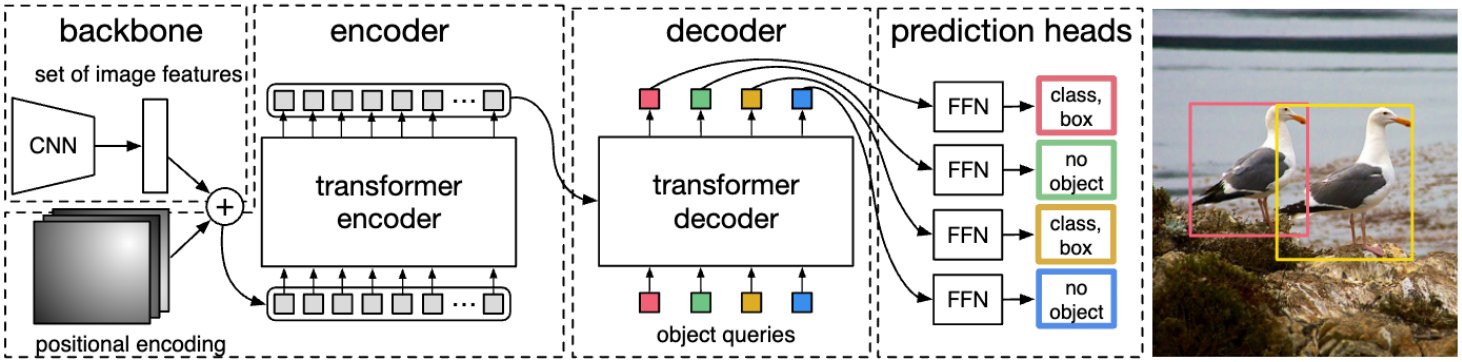



Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

Figure: DETR.

# Number of parameters

If $L$ is the number of layer, $d$ is the hidden size, $V$ is the number of vocabulary.

- $L \times (d \times d * 12 + (4 + 5 + 2 * 2) * d) + V * d$

For example, nanoGPT-small:

- 12*(768*768*12+13*768) + 50000*768 = 123454464

# Memory usage in training

Let $\boldsymbol{X} \in \mathcal{R}^{b \times n \times d}$, $L$ is the number of layer, $h$ is the number of heads.

In the forward process, we need to cache the following variables

- $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{O} \in \mathcal{R}^{b \times n \times d}$
- $\boldsymbol{P}, \boldsymbol{A} \in \mathcal{R}^{b \times n \times n}$
- in each block, we have $h$ times of $\boldsymbol{P}$ and $\boldsymbol{A}$
- in FFN, we have $\mathcal{R}^{b \times n \times 4d}$ (ReLU replace can be used.)
- in residual shortcut, we have two $\mathcal{R}^{b \times n \times d}$

**Note:**

Flash-attention is extremly important in reducing memory cost and speeding up the training.

# Computation complexity

- Self-attention module

$$\mathcal{O}(L \times b \times n \times d \times 4d + L \times b \times n \times n \times d \times 2)$$

- FFN module

$$\mathcal{O}(L \times b \times n \times d \times 8d)$$

# Positional encoding [7]

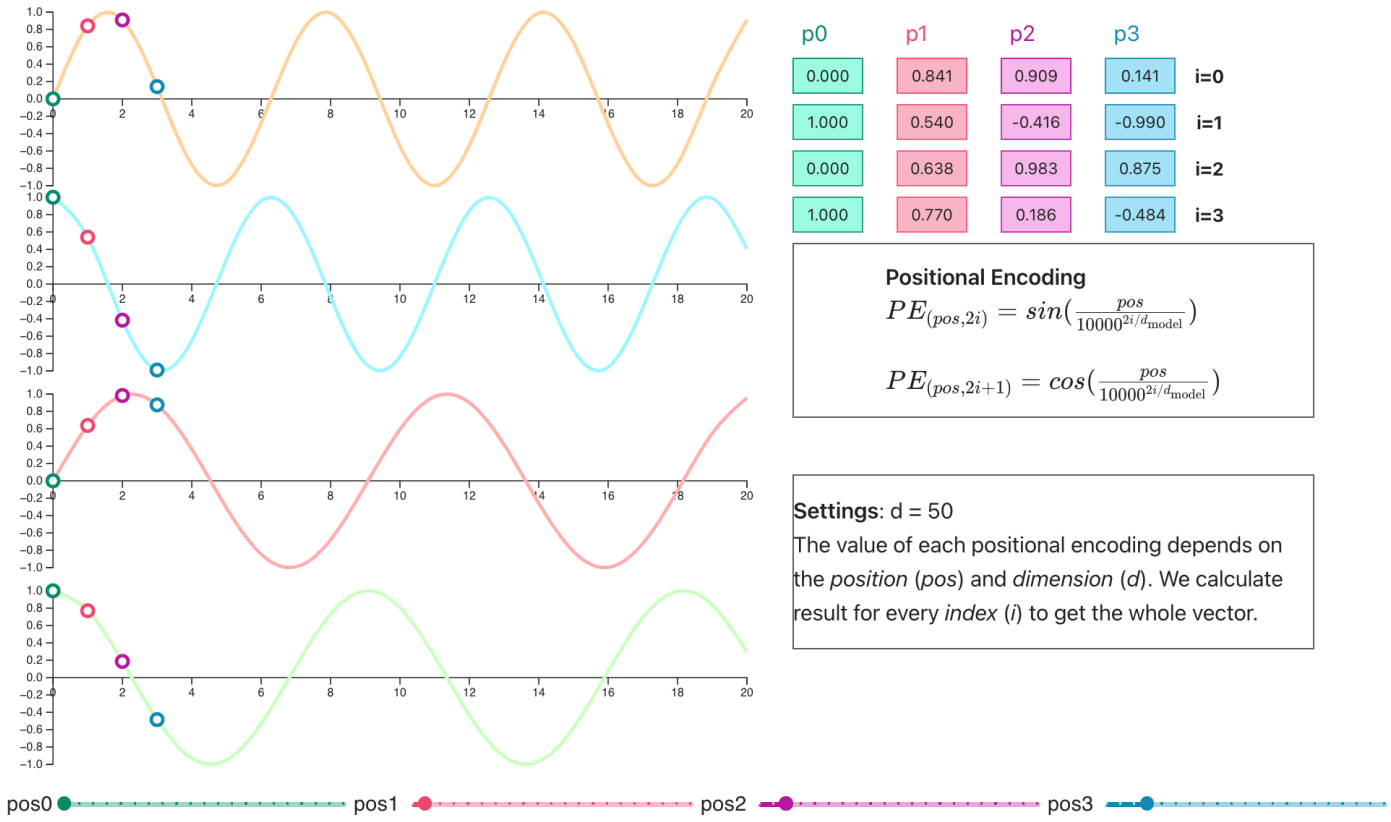
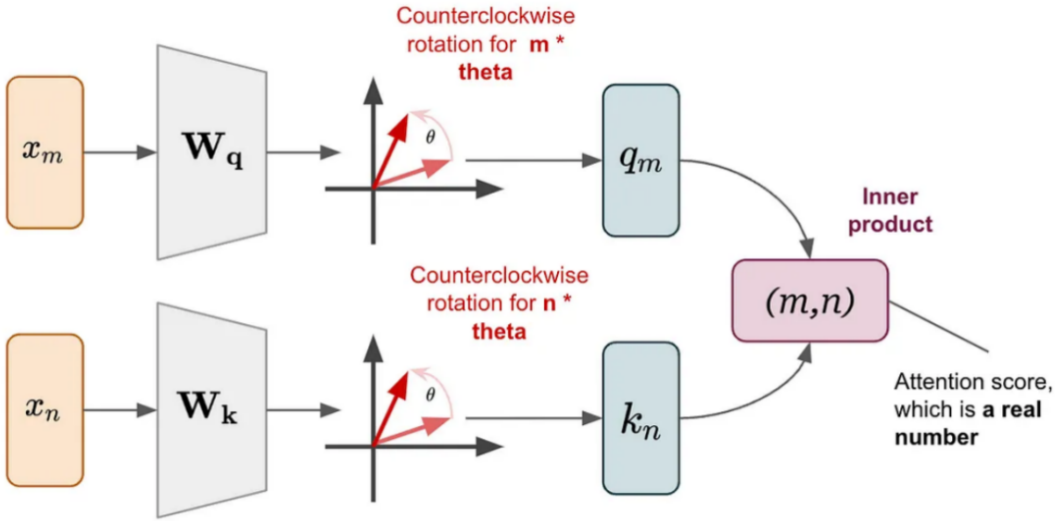



Figure: Visualization of position embedding.

[7] Understanding positional embedding blog

# Rotary position embedding[8]




$$f_q(\boldsymbol{x}_m, m) = (\boldsymbol{W}_q \boldsymbol{x}_m) e^{im\theta}$$
$$f_k(\boldsymbol{x}_n, n) = (\boldsymbol{W}_k \boldsymbol{x}_n) e^{in\theta}$$
$$g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n) = \mathrm{Re}[(\boldsymbol{W}_q \boldsymbol{x}_m)(\boldsymbol{W}_k \boldsymbol{x}_n)^* e^{i(m-n)\theta}]$$

Figure: Visualization of RoPE.

[8] RoPE blog

# Rotary position embedding

$$\boldsymbol{R}_{\Theta,m}^{d} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

And the overall transformation applied to a token embedding is:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}_{\Theta,m}^{d} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

$$\boldsymbol{R}_{\Theta,m}^{d} \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

Figure: Computational process of RoPE.

# Advanced skills

- Flash-attention
- GQA
- Multi-head Latent Attention
- KV-cache
- MoE
- LoRA

# Chain Rule

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^l} &= \frac{\partial \boldsymbol{x}^{l+1}}{\partial \boldsymbol{x}^l} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{l+1}} \\
&= \frac{\partial \boldsymbol{x}^{l+1}}{\partial \boldsymbol{x}^l} \cdots \frac{\partial \boldsymbol{x}^{L}}{\partial \boldsymbol{x}^{L-1}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{L}} \\
&= \frac{\partial \boldsymbol{x}^{l+1}}{\partial \boldsymbol{x}^l} \cdots \frac{\partial \boldsymbol{x}^{L}}{\partial \boldsymbol{x}^{L-1}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}^{L}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{o}} \\
&= \left( \prod_{k=l+1}^{L} \frac{\partial \boldsymbol{x}^k}{\partial \boldsymbol{x}^{k-1}} \right) \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}^{L}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{o}} \\
&= \left( \prod_{k=l+1}^{L} \frac{\partial \boldsymbol{h}^k}{\partial \boldsymbol{x}^{k-1}} \frac{\partial \boldsymbol{r}^k}{\partial \boldsymbol{h}^k} \frac{\partial \boldsymbol{x}^k}{\partial \boldsymbol{r}^k} \right) \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}^{L}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{o}} \\
&= \left( \prod_{k=l+1}^{L} \left(\boldsymbol{W}^k\right)^\top \operatorname{diag}\left(\mathbf{1}\left(\boldsymbol{W}^k \boldsymbol{x}^{k-1} > \mathbf{0}\right)\right) \frac{\sqrt{D}\left(\boldsymbol{I} - \frac{1}{D}\mathbf{1}\mathbf{1}^\top\right)}{\sqrt{\|\boldsymbol{y}^k\|_2^2 + \epsilon}} \left(\boldsymbol{I} - \frac{\boldsymbol{y}^k \boldsymbol{y}^{k\top}}{\|\boldsymbol{y}^k\|_2^2 + \epsilon}\right) \operatorname{diag}\left(\boldsymbol{\gamma}^k\right) \right) \boldsymbol{O}^\top (\boldsymbol{p} - \boldsymbol{t}) .
\end{aligned}
$$

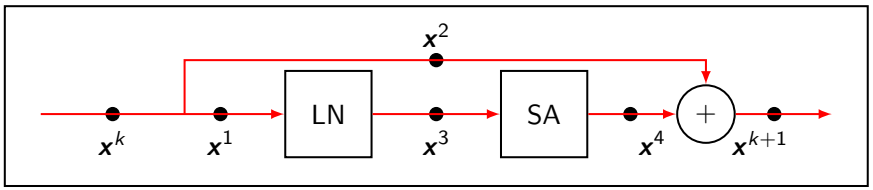



Figure: Forward process of self-attention module

Forward process of Transformer:

$$x^1 = x^2 = x^k,$$
$$x^3 = \mathrm{LN}(x^1),$$
$$x^4 = \mathrm{SA}(x^3),$$
$$x^{k+1} = x^2 + x^4.$$

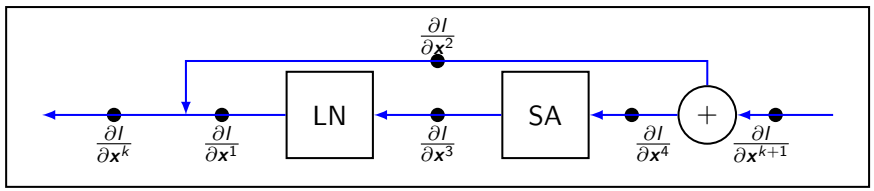



Figure: Backward process of self-attention module

Backward process of Transformer:

$$\begin{aligned}
\frac{\partial l}{\partial \boldsymbol{x}^{k+1}} &= \frac{\partial l}{\partial \boldsymbol{x}^4} = \frac{\partial l}{\partial \boldsymbol{x}^2}, \\
\frac{\partial l}{\partial \boldsymbol{x}^3} &= \frac{\partial l}{\partial \boldsymbol{x}^4} \boldsymbol{J}_{\boldsymbol{x}^4}(\boldsymbol{x}^3), \\
\frac{\partial l}{\partial \boldsymbol{x}^1} &= \frac{\partial l}{\partial \boldsymbol{x}^4} \boldsymbol{J}_{\boldsymbol{x}^4}(\boldsymbol{x}^3), \\
\frac{\partial l}{\partial \boldsymbol{x}^1} &= \frac{\partial l}{\partial \boldsymbol{x}^3} \boldsymbol{J}_{\boldsymbol{x}^3}(\boldsymbol{x}^1), \\
\frac{\partial l}{\partial \boldsymbol{x}^k} &= \frac{\partial l}{\partial \boldsymbol{x}^2} + \frac{\partial l}{\partial \boldsymbol{x}^1},
\end{aligned}$$

$\boldsymbol{J}_{\boldsymbol{x}^3}(\boldsymbol{x}^1)$ is the Jacobian matrix of $\boldsymbol{x}^3$ with respect to $\boldsymbol{x}^1$.

Denote it as $\boldsymbol{J}_{\boldsymbol{x}^3}(\boldsymbol{x}^1) = \frac{\partial \boldsymbol{x}^3}{\partial \boldsymbol{x}^1}$.

# LayerNorm

- Forward Process

$$\mathrm{LN}(\boldsymbol{x}) = \boldsymbol{\gamma} \odot \frac{\sqrt{d}\boldsymbol{y}}{\sqrt{\|\boldsymbol{y}\|_2^2 + \epsilon}} + \boldsymbol{\beta}, \quad \text{and } \boldsymbol{y} = \left(\boldsymbol{I} - \frac{1}{d}\mathbf{1}\mathbf{1}^\top\right)\boldsymbol{x},$$

where $\boldsymbol{x} \in \mathcal{R}^d$

- Backward Process

$$\frac{\partial \mathrm{LN}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial \mathrm{LN}(\boldsymbol{x})}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\sqrt{d}}{\sqrt{\|\boldsymbol{y}\|_2^2 + \epsilon}}\mathrm{Diag}(\boldsymbol{\gamma})\left(\boldsymbol{I} - \frac{\boldsymbol{y}\boldsymbol{y}^\top}{\|\boldsymbol{y}\|_2^2 + \epsilon}\right)\left(\boldsymbol{I} - \frac{1}{d}\mathbf{1}\mathbf{1}^\top\right).$$

$$\frac{\partial \mathrm{LN}(\boldsymbol{x})}{\partial \boldsymbol{\gamma}} = \mathrm{Diag}(\boldsymbol{I} - \frac{\boldsymbol{y}\boldsymbol{y}^\top}{\|\boldsymbol{y}\|_2^2 + \epsilon}), \frac{\partial \mathrm{LN}(\boldsymbol{x})}{\partial \boldsymbol{\beta}} = \frac{\partial \mathrm{LN}(\boldsymbol{x})}{\partial \boldsymbol{x}},$$

$$\frac{\partial \mathrm{LN}(\boldsymbol{x})}{\partial \boldsymbol{y}} = \frac{\sqrt{d}}{\sqrt{\|\boldsymbol{y}\|_2^2 + \epsilon}}\mathrm{Diag}(\boldsymbol{\gamma})\left(\boldsymbol{I} - \frac{\boldsymbol{y}\boldsymbol{y}^\top}{\|\boldsymbol{y}\|_2^2 + \epsilon}\right)$$

# Feed-forward network

- Forward Process

$$\boldsymbol{y} = \mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{W}_2 \max(0, \boldsymbol{W}_1 \boldsymbol{x}),$$

- Backward Process

$$\boldsymbol{J}_y(\boldsymbol{x}) = \frac{\partial \mathrm{FFN}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \boldsymbol{W}_2 \mathrm{Diag}(\mathbf{1}(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1 > \mathbf{0}))\boldsymbol{W}_1$$

# Residual block

- Forward Process

$$\boldsymbol{y} = \boldsymbol{x} + f(\boldsymbol{x})$$

- Backward Process

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{I} + \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$$

# Softmax and cross-entropy loss

- Forward Process

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

$$\mathcal{L} = -\sum_{i=1}^{n} y_i \log(p_i)$$

where $\boldsymbol{y}$ is one-hot label vector

- Backward Process

$$\nabla_{\boldsymbol{x}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\right)^{\top} = \boldsymbol{p} - \boldsymbol{y} = \text{softmax}(\boldsymbol{x}) - \boldsymbol{y}$$

- Note: The gradient simplifies elegantly to $\boldsymbol{p} - \boldsymbol{y}$ due to the properties of softmax and cross-entropy.

# Softmax and cross-entropy loss

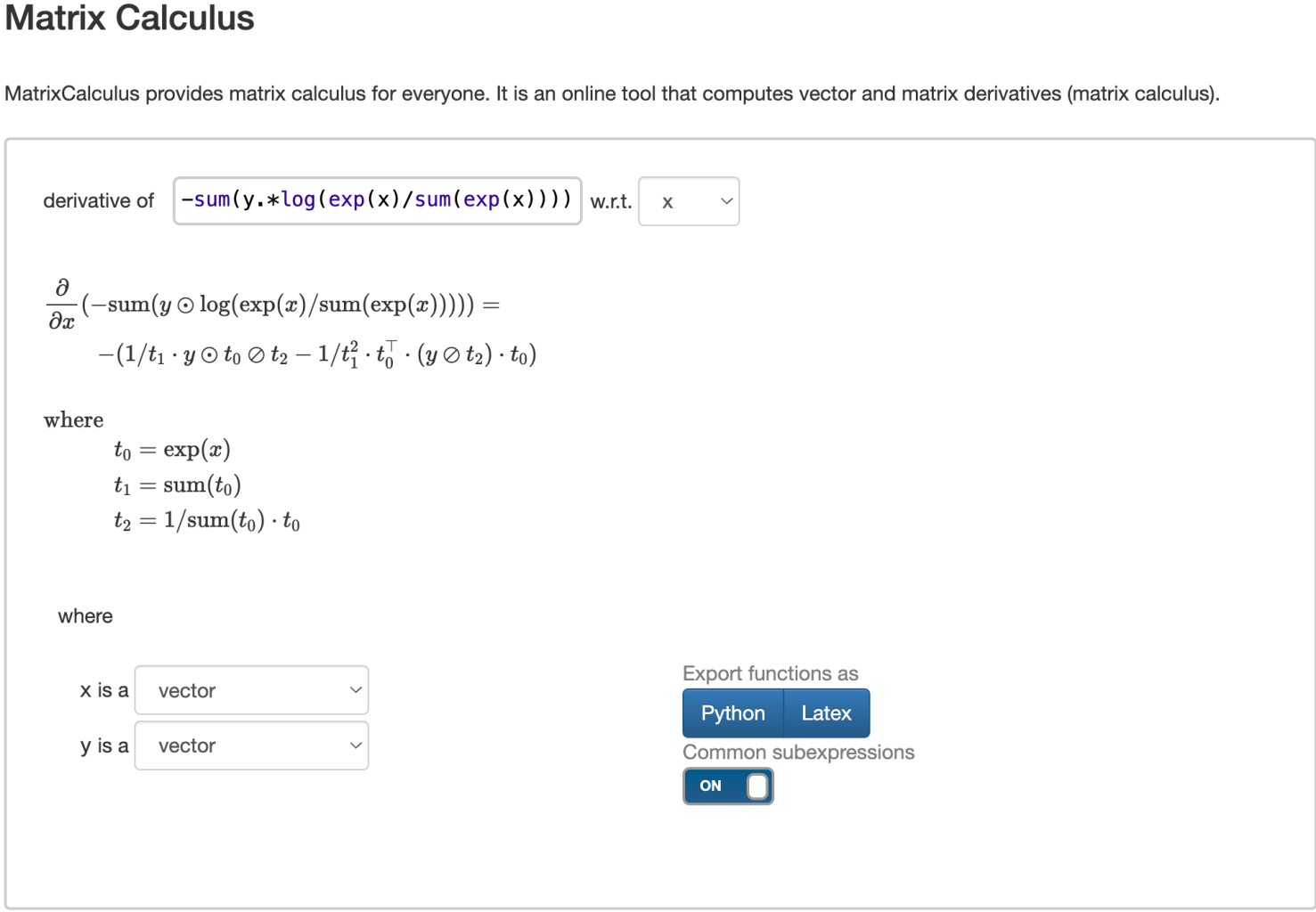



Figure: Gradient of softmax and cross-entropy.

# Self-attention: two forms

- Form 1: if $\boldsymbol{X} \in \mathcal{R}^{d \times n}$,

$$\boldsymbol{Y} = SA(\boldsymbol{X}) = \boldsymbol{W}_v \boldsymbol{X} \, \text{softmax}(\frac{\boldsymbol{X}^\top \boldsymbol{W}_q^\top \boldsymbol{W}_k \boldsymbol{X}}{\sqrt{d/h}})$$

  where softmax is conducted in the row direction, $\boldsymbol{Y} \in \mathcal{R}^{d \times n}$.

- Form 2: if $\boldsymbol{X} \in \mathcal{R}^{n \times d}$,

$$\boldsymbol{Y} = SA(\boldsymbol{X}) = \text{softmax}(\frac{\boldsymbol{X} \boldsymbol{W}_q \boldsymbol{W}_k^\top \boldsymbol{X}^\top}{\sqrt{d/h}}) \boldsymbol{X} \boldsymbol{W}_v$$

  where softmax is conducted in the column direction, $\boldsymbol{Y} \in \mathcal{R}^{n \times d}$.

# Kronecker product of self-attention

$$\text{Let } \boldsymbol{P} = \boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \boldsymbol{X}, \quad \boldsymbol{A} = \text{softmax}(\frac{\boldsymbol{P}}{\sqrt{d_q}}),$$

where $\boldsymbol{A}$ is the attention map, and $\frac{\boldsymbol{P}}{\sqrt{d_q}}$ is usually called as the logit. Assume $\boldsymbol{X} \in \mathcal{R}^{d \times n}$, $\boldsymbol{W}_q \in \mathcal{R}^{d_q \times d}$, $\boldsymbol{W}_k \in \mathcal{R}^{d_q \times d}$, according to vectorization and matrix calculus, we have the following derivations:

$$\frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{W}_q{}^\top \boldsymbol{W}_k)} = \boldsymbol{X}^\top \otimes \boldsymbol{X}^\top, \quad \frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{X})} = (\boldsymbol{X}^\top \boldsymbol{W}_k{}^\top \boldsymbol{W}_q \otimes \boldsymbol{I}_n)\boldsymbol{K} + (\boldsymbol{I}_n \otimes \boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k),$$

$$\frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{W}_q^\top)} = (\boldsymbol{W}_k \boldsymbol{X})^\top \otimes \boldsymbol{X}^\top, \quad \frac{\partial \operatorname{vec}(\boldsymbol{P})}{\partial \operatorname{vec}(\boldsymbol{W}_k)} = \boldsymbol{X}^\top \otimes (\boldsymbol{W}_q \boldsymbol{X})^\top,$$

where $\boldsymbol{I}_n \in \mathcal{R}^{n \times n}$ denotes an identity matrix with shape $n \times n$, $\boldsymbol{K}$ is the commutation matrix, which depends on the dimensions of $\boldsymbol{X}$. Since $\boldsymbol{X} \in \mathcal{R}^{d \times n}$, then we know $\boldsymbol{K} \in \mathcal{R}^{nd \times nd}$. The commutation matrix $\boldsymbol{K}$ has the property that $\operatorname{vec}(\boldsymbol{X}^\top) = \boldsymbol{K} \operatorname{vec}(\boldsymbol{X})$ for any matrix $\boldsymbol{X}$.

# Single-head self-attention

- Forward Process

$$\boldsymbol{Y} = \boldsymbol{W}_v \boldsymbol{X} \boldsymbol{A},$$

$$\text{where } \boldsymbol{P} = \boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \boldsymbol{X}, \quad \boldsymbol{A} = \text{softmax}(\frac{\boldsymbol{P}}{\sqrt{d_q}}).$$

- Backward Process

$$\begin{aligned}\frac{\partial \text{vec}(\boldsymbol{Y})}{\partial \text{vec}(\boldsymbol{X})} &= (\boldsymbol{A}^\top \otimes \boldsymbol{W}_v) + (\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{X})}, \\ &= (\boldsymbol{A}^\top \otimes \boldsymbol{W}_v) + (\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{P})} \frac{\partial \text{vec}(\boldsymbol{P})}{\partial \text{vec}(\boldsymbol{X})}.\end{aligned}$$

# Kronecker product of softmax operator

Define

$$\boldsymbol{B} = \exp(\frac{\boldsymbol{P}}{\sqrt{d_q}}), \quad \boldsymbol{C} = \mathbf{1}\mathbf{1}^\top \exp(\frac{\boldsymbol{P}}{\sqrt{d_q}}) = \mathbf{1}\mathbf{1}^\top \boldsymbol{B}.$$

In this way, we can represent the attention matrix $\boldsymbol{A}$ as $\boldsymbol{A} = \boldsymbol{B} \oslash \boldsymbol{C}$. Then, we can vectorize the equation,

$$\text{vec}(\boldsymbol{A}) = \text{vec}(\boldsymbol{B} \oslash \boldsymbol{C}) = \text{vec}(\boldsymbol{B}) \oslash \text{vec}(\boldsymbol{C}).$$

According to the chain rule, we have

$$\frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{P})} = \frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{B})} \frac{\partial \text{vec}(\boldsymbol{B})}{\partial \text{vec}(\boldsymbol{P})} + \frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{C})} \frac{\partial \text{vec}(\boldsymbol{C})}{\partial \text{vec}(\boldsymbol{P})}.$$

# Kronecker product of softmax operator

Let us calculate each individual term. We have

$$\begin{aligned}
\frac{\partial \text{vec}(\boldsymbol{A})}{\partial\, \text{vec}(\boldsymbol{B})} &= \mathbf{1}_{n^2} \oslash \text{Diag}(\text{vec}(\boldsymbol{C})), \\
\frac{\partial \text{vec}(\boldsymbol{B})}{\partial \text{vec}(\boldsymbol{P})} &= \frac{\text{Diag}(\text{vec}(\boldsymbol{B}))}{\sqrt{d_q}}, \\
\frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{C})} &= -\text{Diag}(\text{vec}(\boldsymbol{B}) \oslash (\text{vec}(\boldsymbol{C}) \odot \text{vec}(\boldsymbol{C}))), \\
\frac{\partial \text{vec}(\boldsymbol{C})}{\partial \text{vec}(\boldsymbol{P})} &= \frac{(\boldsymbol{I}_n \otimes \mathbf{1}_n \mathbf{1}_n^\top)\text{Diag}(\text{vec}(\boldsymbol{B}))}{\sqrt{d_q}},
\end{aligned}$$

where $\boldsymbol{I}_n$ is a $n \times n$ identity matrix and $\otimes$ is the Kronecker product, $\mathbf{1}_{nn}$ denotes a $\mathcal{R}^{n^2}$ all one vector.

# Kronecker product of softmax operator

Substitute these four term into the chain rule, we have

$$\begin{aligned}\frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{P})} &= \frac{\left(\mathbf{1}_{n^2} \oslash \text{diag}(\text{vec}(\boldsymbol{C}))\right)\text{diag}(\text{vec}(\boldsymbol{B})) - \text{diag}(\text{vec}(\boldsymbol{B}) \oslash (\text{vec}(\boldsymbol{C}) \odot \text{vec}(\boldsymbol{C})))(\boldsymbol{I}_n \otimes \mathbf{1}_n\mathbf{1}_n^\top)\text{diag}(\text{vec}(\boldsymbol{B}))}{\sqrt{d_q}} \\ &= \frac{\text{diag}(\text{vec}(\boldsymbol{A})) - \text{diag}(\text{vec}(\boldsymbol{B}) \oslash (\text{vec}(\boldsymbol{C}) \odot \text{vec}(\boldsymbol{C})))(\boldsymbol{I}_n \otimes \mathbf{1}_n\mathbf{1}_n^\top)\text{diag}(\text{vec}(\boldsymbol{B}))}{\sqrt{d_q}} \\ &= \frac{\text{blockdiag}(\text{diag}(\boldsymbol{A}_{:,1}) - \boldsymbol{A}_{:,1}\boldsymbol{A}_{:,1}^\top, \ldots, \text{diag}(\boldsymbol{A}_{:,n}) - \boldsymbol{A}_{:,n}\boldsymbol{A}_{:,n}^\top)}{\sqrt{d_q}}.\end{aligned}$$

If $\boldsymbol{A}$ and $\boldsymbol{P}$ are two vectors, if we use $\boldsymbol{a}$ and $\boldsymbol{p}$ denote them individually, then we know

$$\frac{\partial \text{vec}(\boldsymbol{a})}{\partial \text{vec}(\boldsymbol{p})} = \frac{\text{Diag}(\boldsymbol{a}) - \boldsymbol{a}\boldsymbol{a}^\top}{\sqrt{d_q}}.$$

If $\boldsymbol{a}$ approaches to a unit vector $\boldsymbol{e}$, then the Jacobian matrix $\frac{\partial \text{vec}(\boldsymbol{a})}{\partial \text{vec}(\boldsymbol{p})}$ will tend to $\mathbf{0}$.

# Single-head self-attention

- Backward Process

$$\begin{aligned}\frac{\partial \text{vec}(\boldsymbol{Y})}{\partial \text{vec}(\boldsymbol{X})} &= (\boldsymbol{A}^\top \otimes \boldsymbol{W}_v) + (\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{X})}, \\ &= (\boldsymbol{A}^\top \otimes \boldsymbol{W}_v) + (\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\partial \text{vec}(\boldsymbol{A})}{\partial \text{vec}(\boldsymbol{P})} \frac{\partial \text{vec}(\boldsymbol{P})}{\partial \text{vec}(\boldsymbol{X})}.\end{aligned}$$

- the final Jacobian matrix

$$\frac{\partial \text{vec}(\boldsymbol{Y})}{\partial \text{vec}(\boldsymbol{X})} = (\boldsymbol{A}^\top \otimes \boldsymbol{W}_v) + (\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\boldsymbol{J}}{\sqrt{d_q}} \left( (\boldsymbol{X}^\top \boldsymbol{W}_k{}^\top \boldsymbol{W}_q \otimes \boldsymbol{I}_n) \boldsymbol{K} + (\boldsymbol{I}_n \otimes \boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k) \right).$$

where

$$\boldsymbol{J} = \text{blockdiag}(\text{Diag}(\boldsymbol{A}_{:,1}) - \boldsymbol{A}_{:,1} \boldsymbol{A}_{:,1}^\top, \ldots, \text{Diag}(\boldsymbol{A}_{:,n}) - \boldsymbol{A}_{:,n} \boldsymbol{A}_{:,n}^\top).$$

# Single-head self-attention

Let us analysis each term

$$\frac{\partial \text{vec}(\boldsymbol{Y})}{\partial \text{vec}(\boldsymbol{X})} = (\boldsymbol{A}^\top \otimes \boldsymbol{W}_v) + (\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\boldsymbol{J}}{\sqrt{d_q}} \left( (\boldsymbol{X}^\top \boldsymbol{W}_k{}^\top \boldsymbol{W}_q \otimes \boldsymbol{I}_n) \boldsymbol{K} + (\boldsymbol{I}_n \otimes \boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k) \right).$$

What we see from the equation

- we have $\frac{\boldsymbol{J}}{\sqrt{d_q}}$
- we have $\boldsymbol{W}_q{}^\top \boldsymbol{W}_k$
- we even have $(\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\boldsymbol{J}}{\sqrt{d_q}} (\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n)$

**Question:**

what are the singular values of $\sigma \left( (\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\boldsymbol{J}}{\sqrt{d_q}} (\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n) \right)$ ?

# Single-head self-attention

Let us analysis each term in $\sigma\left((\boldsymbol{I}_n \otimes \boldsymbol{W}_v\boldsymbol{X})\frac{\boldsymbol{J}}{\sqrt{d_q}}(\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n)\right)$

- $\boldsymbol{W}_q{}^\top \boldsymbol{W}_k \in \mathcal{R}^{d\times d}$
- $(\boldsymbol{I}_n \otimes \boldsymbol{W}_v\boldsymbol{X}) \in \mathcal{R}^{nd_v\times nn}$
- $\boldsymbol{J} \in \mathcal{R}^{nn\times nn}$
- $(\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n) \in \mathcal{R}^{nn\times nd}$
- $\left((\boldsymbol{I}_n \otimes \boldsymbol{W}_v\boldsymbol{X})\frac{\boldsymbol{J}}{\sqrt{d_q}}(\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n)\right) \in \mathcal{R}^{nd_v\times nd}$

## Questions:

1. $\sigma\left(\boldsymbol{X}\boldsymbol{X}^\top\right)$
2. $\sigma\left(\boldsymbol{W}_q{}^\top \boldsymbol{W}_k\right)$
3. $\sigma\left(\boldsymbol{W}_v\right)$

# Single-head self-attention

Our answers to some important questions in Transformer:

- 1. why Transformer is hard to train? NAN or INF
- 2. why normalization, $\boldsymbol{X}$, is important in Transformer?
- 3. why prenorm is more stable than postnorm? the norm of $\boldsymbol{X}$.
- 4. why deeper layer, $L$, is harder to train?
- 5. why larger hidden dimension, $d$, is harder to train?
- 6. why longer context, $n$, is harder to train?
- 7. why $\sigma(\boldsymbol{W}_q{}^\top \boldsymbol{W}_k)$ is important for model crash?
- 8. why learning rate warmup is important for classical Transformer?
- 9. why backward process always trigger problem in Transformer?
- 10. do we have solutions to so many why? Yes
- more...

# 1. why Transformer is hard to train?

**Reason:**

$\sigma\left((\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\boldsymbol{J}}{\sqrt{d_q}} (\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n)\right)$ is prone to be very large.

# 2. why normalization is important in Transformer?

**Reason:**

because after normalization on $\boldsymbol{X}$, we can always promise $\|\boldsymbol{x}\|_2 = \sqrt{d}$ before $\gamma, \beta$, thus it will restrict $\sigma\left((\boldsymbol{I}_n \otimes \boldsymbol{W}_v\boldsymbol{X})\frac{\boldsymbol{J}}{\sqrt{d_q}}(\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n)\right)$ to be very large.

# 3. why prenorm is more stable than postnorm in Transformer?

**Reason:**

because in postnorm, $\boldsymbol{Y} = \boldsymbol{X} + \mathrm{LN}(\mathrm{sa}(\boldsymbol{X}))$, the norm of $\boldsymbol{y}$ that is a column in $\boldsymbol{Y}$ is always larger than $\boldsymbol{x}$ (because in a high-dimensional space, two vectors are prone to orthgonal to each other), thus, with along the forward process, the norm of $\boldsymbol{x}$ inputted into a self-attention module increases along with the increase of layer, in the backward process, it will triger larger $\sigma\left((\boldsymbol{I}_n \otimes \boldsymbol{W}_v\boldsymbol{X})\frac{\boldsymbol{J}}{\sqrt{d_q}}(\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n)\right)$.

- Derive the joint Jacobian matrix of LN + SA?
- Derive the joint Jacobian matrix of SA + LN?

# 4. why deeper layer is harder to train?

**Reason:**

because deeper layer means we will have more multiples of Jacobian matrix in chain rule, the backward process is more prone to have larger gradients.

# 5. why larger hidden dimension is harder to train?

**Reason:**

because larger $d$ will be prone to lead to larger $\sigma(\boldsymbol{X}\boldsymbol{X}^\top)$ and larger $\boldsymbol{W}_q^\top \boldsymbol{W}_k$.

# 6. why longer context is harder to train?

**Reason:**

because larger $n$ will be prone to lead to larger $\sigma(\boldsymbol{X}\boldsymbol{X}^\top)$.

# 7. why $\sigma(\boldsymbol{W}_q{}^\top \boldsymbol{W}_k)$ is important for model crash?

**Reason:**

because larger $\sigma(\boldsymbol{W}_q{}^\top \boldsymbol{W}_k)$ will be prone to lead to a larger $\sigma\left((\boldsymbol{I}_n \otimes \boldsymbol{W}_v \boldsymbol{X}) \frac{\boldsymbol{J}}{\sqrt{d_q}} (\boldsymbol{X}^\top \boldsymbol{W}_q{}^\top \boldsymbol{W}_k \otimes \boldsymbol{I}_n)\right)$.

# 8. why learning rate warmup is important for classical Transformer?

**Reason:**

because learning rate warmup does not smooth ${\boldsymbol{W}_q}^\top \boldsymbol{W}_k$, but also smooth $\boldsymbol{X}$.

# 9. why backward process always triggers problem in classical Transformer

**Reason:**

because backward process is prone to lead to larger range of value, but in the forward process, we have normalization to constraint the valeus.

# 10. do we have solutions to so many why

**Answer:**

- constraint the singular values of $\boldsymbol{X}$,
- constraint the singular values of $\boldsymbol{W}_q^\top \boldsymbol{W}_k$

# Simulate gradient of self-attention

```python
import torch
import torch.nn.functional as F
import numpy as np

def self_attention_with_grad(X, d_model=512):
    """
    Compute self-attention and its gradient with respect to input X
    Args:
        X: Input tensor of shape (batch_size, seq_len, d_model)
        d_model: Dimension of the model
    Returns:
        Y: Output tensor
        dY_dX: Gradient of Y with respect to X
    """
    # Ensure X requires gradient and create a fresh copy
    X = X.clone().detach().requires_grad_(True)

    batch_size, seq_len, _ = X.shape

    # Initialize learnable matrices
    W_q = torch.randn(d_model, d_model, device=X.device)*10
    W_k = torch.randn(d_model, d_model, device=X.device)*10
    W_v = torch.randn(d_model, d_model, device=X.device)

    # Compute Q, K, V
    Q = torch.matmul(X, W_q)  # (batch_size, seq_len, d_model)
    K = torch.matmul(X, W_k)  # (batch_size, seq_len, d_model)
    V = torch.matmul(X, W_v)  # (batch_size, seq_len, d_model)

    # Scaled dot-product attention
    scale = torch.sqrt(torch.tensor(d_model, dtype=torch.float32, device=X.device))
    attention_scores = torch.matmul(Q, K.transpose(-2, -1)) / scale  # (batch_size, seq_len, seq_len)
    attention_probs = F.softmax(attention_scores, dim=-1)

    # Compute output
    Y = torch.matmul(attention_probs, V)  # (batch_size, seq_len, d_model)

    # Create a random gradient tensor to backpropagate
    grad_output = torch.randn_like(Y)

    # Compute gradient
    Y.backward(grad_output)

    # Get the gradient and make sure it exists
    assert X.grad is not None, "Gradient computation failed"
    grad = X.grad.clone()

    return Y.detach(), grad
```

Figure: Simulate gradient of self-attention

```python
def main():
    # Set random seed for reproducibility
    torch.manual_seed(42)

    # Create sample input
    batch_size = 2
    seq_len = 1000
    d_model = 768
    X = torch.randn(batch_size, seq_len, d_model)

    # Compute self-attention and its gradient
    Y, X_gradient = self_attention_with_grad(X, d_model=d_model)

    # Print norms for verification
    print(f"Output Y norm: {Y.norm().item():.4f}")
    print(f"Gradient dY/dX norm: {X_gradient.norm().item():.4f}")

    # Print shapes for verification
    print(f"\nShapes:")
    print(f"Input X: {X.shape}")
    print(f"Output Y: {Y.shape}")
    print(f"Gradient dY/dX: {X_gradient.shape}")

if __name__ == "__main__":
    main()
```

```
Output Y norm: 34595.2461
Gradient dY/dX norm: 42357236.0000

Shapes:
Input X: torch.Size([2, 1000, 768])
Output Y: torch.Size([2, 1000, 768])
Gradient dY/dX: torch.Size([2, 1000, 768])
```

Figure: Simulate gradient of self-attention.

# Simulate gradient of FFN

```python
import torch
import torch.nn.functional as F
import numpy as np

def ffn_with_grad(X, d_model=512, d_ff=2048):
    """
    Compute FFN (Feed-Forward Network) and its gradient with respect to input X
    Args:
        X: Input tensor of shape (batch_size, seq_len, d_model)
        d_model: Model dimension
        d_ff: Feed-forward dimension (usually 4*d_model)
    Returns:
        Y: Output tensor
        dL_dX: Gradient of L with respect to X
    """
    # Ensure X requires gradient and create a fresh copy
    X = X.clone().detach().requires_grad_(True)

    # Initialize weights for the two linear transformations
    W1 = torch.randn(d_model, d_ff, device=X.device) #/ np.sqrt(d_model)
    b1 = torch.zeros(d_ff, device=X.device)

    W2 = torch.randn(d_ff, d_model, device=X.device) #/ np.sqrt(d_ff)
    b2 = torch.zeros(d_model, device=X.device)

    # First linear transformation and ReLU
    hidden = torch.matmul(X, W1) + b1  # (batch_size, seq_len, d_ff)
    hidden = F.relu(hidden)

    # Second linear transformation
    Y = torch.matmul(hidden, W2) + b2  # (batch_size, seq_len, d_model)

    # Create a random gradient tensor to backpropagate
    grad_output = torch.randn_like(Y)
    print(f"Input grad_output norm: {grad_output.norm().item():.4f}")


    # Compute gradient
    Y.backward(grad_output)

    # Get the gradient and make sure it exists
    assert X.grad is not None, "Gradient computation failed"
    grad = X.grad.clone()

    return Y.detach(), grad
```

Figure: Simulate gradient of FFN.

```python
def main():
    # Set random seed for reproducibility
    torch.manual_seed(42)

    # Create sample input
    batch_size = 2
    seq_len = 1000
    d_model = 768
    d_ff = 768*4  # Usually 4*d_model

    X = torch.randn(batch_size, seq_len, d_model)

    # Compute FFN and its gradient
    Y, X_gradient = ffn_with_grad(X, d_model=d_model, d_ff=d_ff)

    # Print norms for verification
    print(f"Input X norm: {X.norm().item():.4f}")
    print(f"Output Y norm: {Y.norm().item():.4f}")
    print(f"Gradient dY/dX norm: {X_gradient.norm().item():.4f}")

    # Verify gradient shape matches input shape
    assert X.shape == X_gradient.shape, "Gradient shape mismatch!"

if __name__ == "__main__":
    main()
```

```
Input grad_output norm: 1239.5171
Input X norm: 1239.5165
Output Y norm: 1342592.1250
Gradient dY/dX norm: 1346009.6250
```

Figure: Simulate gradient of FFN.

# Reproduce NaN problem

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class MultiHeadSelfAttention(nn.Module):
    def __init__(self, hidden_dim, num_heads):
        super(MultiHeadSelfAttention, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_heads = num_heads
        self.head_dim = hidden_dim // num_heads
        scale = 1000

        rank = 2
        self.rank = rank

        # Low-rank decomposition for attention weights
        self.Aq = nn.Parameter(scale * torch.randn(hidden_dim, rank) / torch.sqrt(torch.tensor(hidden_dim + rank, dtype=torch.float32)))
        self.Bq = nn.Parameter(scale * torch.randn(rank, hidden_dim) / torch.sqrt(torch.tensor(rank + hidden_dim, dtype=torch.float32)))
        self.Ak = nn.Parameter(scale * torch.randn(hidden_dim, rank) / torch.sqrt(torch.tensor(hidden_dim + rank, dtype=torch.float32)))
        self.Bk = nn.Parameter(scale * torch.randn(rank, hidden_dim) / torch.sqrt(torch.tensor(rank + hidden_dim, dtype=torch.float32)))
        self.Av = nn.Parameter(scale * torch.randn(hidden_dim, rank) / torch.sqrt(torch.tensor(hidden_dim + rank, dtype=torch.float32)))
        self.Bv = nn.Parameter(scale * torch.randn(rank, hidden_dim) / torch.sqrt(torch.tensor(rank + hidden_dim, dtype=torch.float32)))
        self.Ao = nn.Parameter(scale * torch.randn(hidden_dim, rank) / torch.sqrt(torch.tensor(hidden_dim + rank, dtype=torch.float32)))
        self.Bo = nn.Parameter(scale * torch.randn(rank, hidden_dim) / torch.sqrt(torch.tensor(rank + hidden_dim, dtype=torch.float32)))
        self.Wo = nn.Parameter(scale * torch.randn(hidden_dim, hidden_dim) / torch.sqrt(torch.tensor(hidden_dim + hidden_dim, dtype=torch.float32)))

    def scaled_dot_product_attention(self, Q, K, V):
        scores = torch.matmul(Q, K.transpose(-2, -1)) / (self.head_dim ** 0.5)
        attn_weights = F.softmax(scores, dim=-1)
        attn_output = torch.matmul(attn_weights, V)
        return attn_output

    def forward(self, x):
        # Compute low-rank Q, K, V
        Q = torch.matmul(torch.matmul(x, self.Aq), self.Bq)  # Low-rank Wq
        K = torch.matmul(torch.matmul(x, self.Ak), self.Bk)  # Low-rank Wk
        V = torch.matmul(torch.matmul(x, self.Av), self.Bv)  # Low-rank Wv

        # Reshape for multi-head attention
        Q = Q.view(Q.shape[0], Q.shape[1], self.num_heads, self.head_dim).permute(0, 2, 1, 3)
        K = K.view(K.shape[0], K.shape[1], self.num_heads, self.head_dim).permute(0, 2, 1, 3)
        V = V.view(V.shape[0], V.shape[1], self.num_heads, self.head_dim).permute(0, 2, 1, 3)

        # Scaled dot-product attention
        attn_output = self.scaled_dot_product_attention(Q, K, V)
        attn_output = attn_output.permute(0, 2, 1, 3).contiguous()
        attn_output = attn_output.view(attn_output.shape[0], attn_output.shape[1], -1)

        # Output projection
        attn_output = torch.matmul(attn_output, self.Wo)
        return attn_output
```

Figure: Reproduce NaN problem (part 1).

```
+ Code  + Text
```

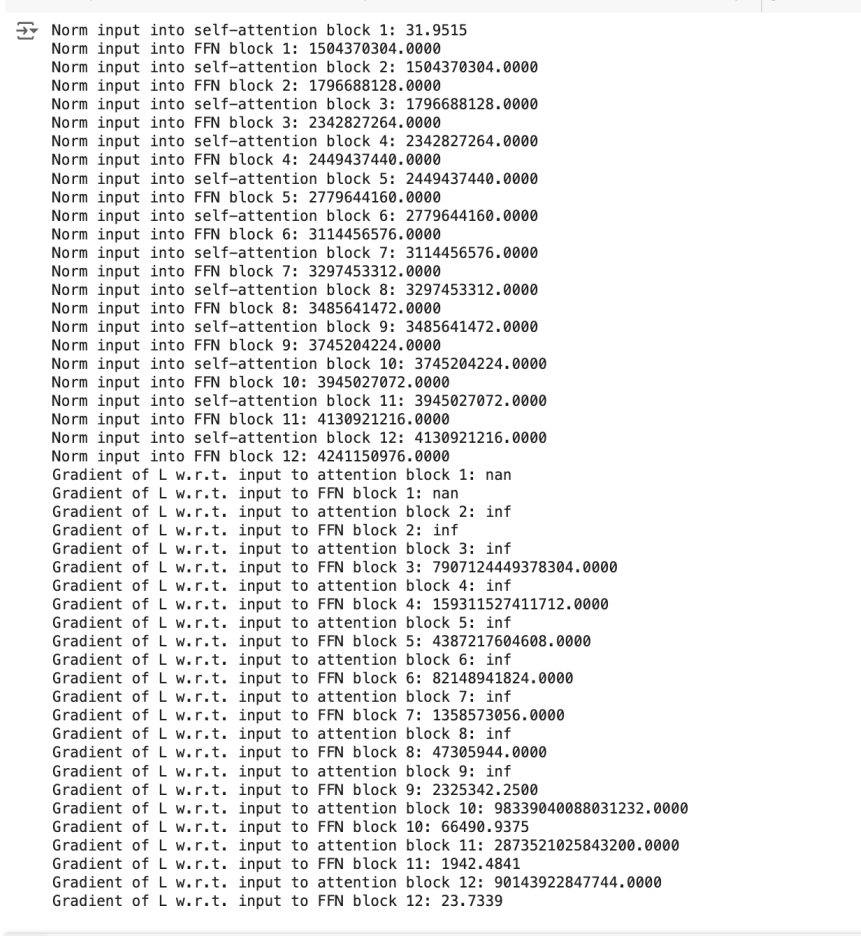
```python
class FeedForwardNetwork(nn.Module):
    def __init__(self, hidden_dim, ff_dim):
        super(FeedForwardNetwork, self).__init__()
        scale = 0.1
        self.W1 = nn.Parameter(scale*torch.randn(hidden_dim, ff_dim) / torch.sqrt(torch.tensor(hidden_dim + ff_dim, dtype=torch.float32)))
        self.W2 = nn.Parameter(scale*torch.randn(ff_dim, hidden_dim) / torch.sqrt(torch.tensor(hidden_dim + ff_dim, dtype=torch.float32)))

    def forward(self, x):
        ffn_output = torch.matmul(x, self.W1)
        ffn_output = F.relu(ffn_output)
        ffn_output = torch.matmul(ffn_output, self.W2)
        return ffn_output

class PreNormTransformerBlock(nn.Module):
    def __init__(self, hidden_dim, num_heads, ff_dim):
        super(PreNormTransformerBlock, self).__init__()
        self.attention = MultiHeadSelfAttention(hidden_dim, num_heads)
        self.ffn = FeedForwardNetwork(hidden_dim, ff_dim)
        self.norm1 = nn.LayerNorm(hidden_dim)
        self.norm2 = nn.LayerNorm(hidden_dim)

    def forward(self, x):
        # Pre-norm and attention
        x1 = x
        x_norm1 = self.norm1(x1)
        x_norm1.retain_grad()  # Retain gradients for input to attention
        attn_output = self.attention(x_norm1)
        x2 = x1 + attn_output

        # Pre-norm and feed-forward network
        x_norm2 = self.norm2(x2)
        x_norm2.retain_grad()  # Retain gradients for input to FFN
        ffn_output = self.ffn(x_norm2)
        x = x2 + ffn_output
        return x, x1, x2, x_norm1, x_norm2

class PreNormTransformerModel(nn.Module):
    def __init__(self, num_layers, hidden_dim, num_heads, ff_dim):
        super(PreNormTransformerModel, self).__init__()
        self.layers = nn.ModuleList([PreNormTransformerBlock(hidden_dim, num_heads, ff_dim) for _ in range(num_layers)])
        self.attention_inputs = []  # To store inputs to attention blocks
        self.ffn_inputs = []  # To store inputs to FFN blocks

    def forward(self, x):
        self.attention_inputs = []
        self.ffn_inputs = []
        for i, layer in enumerate(self.layers):
            x, x1, x2, x_norm1, x_norm2 = layer(x)

            norm = torch.norm(x1, dim=-1).mean()  # Calculate the mean norm of activations
            print(f"Norm input into self-attention block {i + 1}: {norm.item():.4f}")

            norm = torch.norm(x2, dim=-1).mean()  # Calculate the mean norm of activations
            print(f"Norm input into FFN block {i + 1}: {norm.item():.4f}")

            self.attention_inputs.append(x_norm1)
            self.ffn_inputs.append(x_norm2)
        return x
```

Figure: Reproduce NaN problem (part 2).

# Reproduce NaN problem

```
+ Code + Text

# Define input tensor and model parameters
batch_size = 1
seq_len = 1000
hidden_dim = 1024
num_layers = 12
num_heads = 8
ff_dim = 4096

# Instantiate model
model = PreNormTransformerModel(num_layers, hidden_dim, num_heads, ff_dim)

# Random input tensor
X = 1*torch.randn(batch_size, seq_len, hidden_dim, requires_grad=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
X = X.to(device)

# Define a dummy loss function
dummy_target = torch.ones(batch_size, seq_len, hidden_dim).to(device)
criterion = nn.MSELoss()

# Forward pass
output = model(X)

# Compute loss
loss = criterion(output, dummy_target)

# Backward pass
loss.backward()

# Output gradients for inputs to attention and FFN blocks
for l, (attn_input, ffn_input) in enumerate(zip(model.attention_inputs, model.ffn_inputs)):
    print(f"Gradient of L w.r.t. input to attention block {l+1}: {torch.mean(attn_input.grad.norm(dim=2)).item():.4f}")
    print(f"Gradient of L w.r.t. input to FFN block {l+1}: {torch.mean(ffn_input.grad.norm(dim=2)).item():.4f}")
```

Figure: Reproduce NaN problem (part 3).



```
Norm input into self-attention block 1: 31.9515
Norm input into FFN block 1: 1504370304.0000
Norm input into self-attention block 2: 1504370304.0000
Norm input into FFN block 2: 1796688128.0000
Norm input into self-attention block 3: 1796688128.0000
Norm input into FFN block 3: 2342827264.0000
Norm input into self-attention block 4: 2342827264.0000
Norm input into FFN block 4: 2449437440.0000
Norm input into self-attention block 5: 2449437440.0000
Norm input into FFN block 5: 2779644160.0000
Norm input into self-attention block 6: 2779644160.0000
Norm input into FFN block 6: 3114456576.0000
Norm input into self-attention block 7: 3114456576.0000
Norm input into FFN block 7: 3297453312.0000
Norm input into self-attention block 8: 3297453312.0000
Norm input into FFN block 8: 3485641472.0000
Norm input into self-attention block 9: 3485641472.0000
Norm input into FFN block 9: 3745204224.0000
Norm input into self-attention block 10: 3745204224.0000
Norm input into FFN block 10: 3945027072.0000
Norm input into self-attention block 11: 3945027072.0000
Norm input into FFN block 11: 4130921216.0000
Norm input into self-attention block 12: 4130921216.0000
Norm input into FFN block 12: 4241150976.0000
Gradient of L w.r.t. input to attention block 1: nan
Gradient of L w.r.t. input to FFN block 1: nan
Gradient of L w.r.t. input to attention block 2: inf
Gradient of L w.r.t. input to FFN block 2: inf
Gradient of L w.r.t. input to attention block 3: inf
Gradient of L w.r.t. input to FFN block 3: 7907124449378304.0000
Gradient of L w.r.t. input to attention block 4: inf
Gradient of L w.r.t. input to FFN block 4: 159311527411712.0000
Gradient of L w.r.t. input to attention block 5: inf
Gradient of L w.r.t. input to FFN block 5: 4387217604608.0000
Gradient of L w.r.t. input to attention block 6: inf
Gradient of L w.r.t. input to FFN block 6: 82148941824.0000
Gradient of L w.r.t. input to attention block 7: inf
Gradient of L w.r.t. input to FFN block 7: 1358573056.0000
Gradient of L w.r.t. input to attention block 8: inf
Gradient of L w.r.t. input to FFN block 8: 47305944.0000
Gradient of L w.r.t. input to attention block 9: inf
Gradient of L w.r.t. input to FFN block 9: 2325342.2500
Gradient of L w.r.t. input to attention block 10: 98339040088031232.0000
Gradient of L w.r.t. input to FFN block 10: 66490.9375
Gradient of L w.r.t. input to attention block 11: 2873521025843200.0000
Gradient of L w.r.t. input to FFN block 11: 1942.4841
Gradient of L w.r.t. input to attention block 12: 90143922847744.0000
Gradient of L w.r.t. input to FFN block 12: 23.7339
```

Figure: Reproduce NaN problem (part 4).

# RMSNorm

- Forward Process

$$\text{RMSN}(\boldsymbol{x}) = \boldsymbol{\gamma} \odot \frac{\sqrt{d}\boldsymbol{x}}{\sqrt{\|\boldsymbol{x}\|_2^2 + \epsilon}}$$

where $\boldsymbol{x} \in \mathcal{R}^d$

- Backward Process

$$\frac{\partial \text{RMSN}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\sqrt{d}}{\sqrt{\|\boldsymbol{x}\|_2^2 + \epsilon}} \left( \boldsymbol{I} - \frac{\boldsymbol{x}\boldsymbol{x}^\top}{\|\boldsymbol{x}\|_2^2 + \epsilon} \right) \text{Diag}(\boldsymbol{\gamma}).$$

# RMSNorm gradient

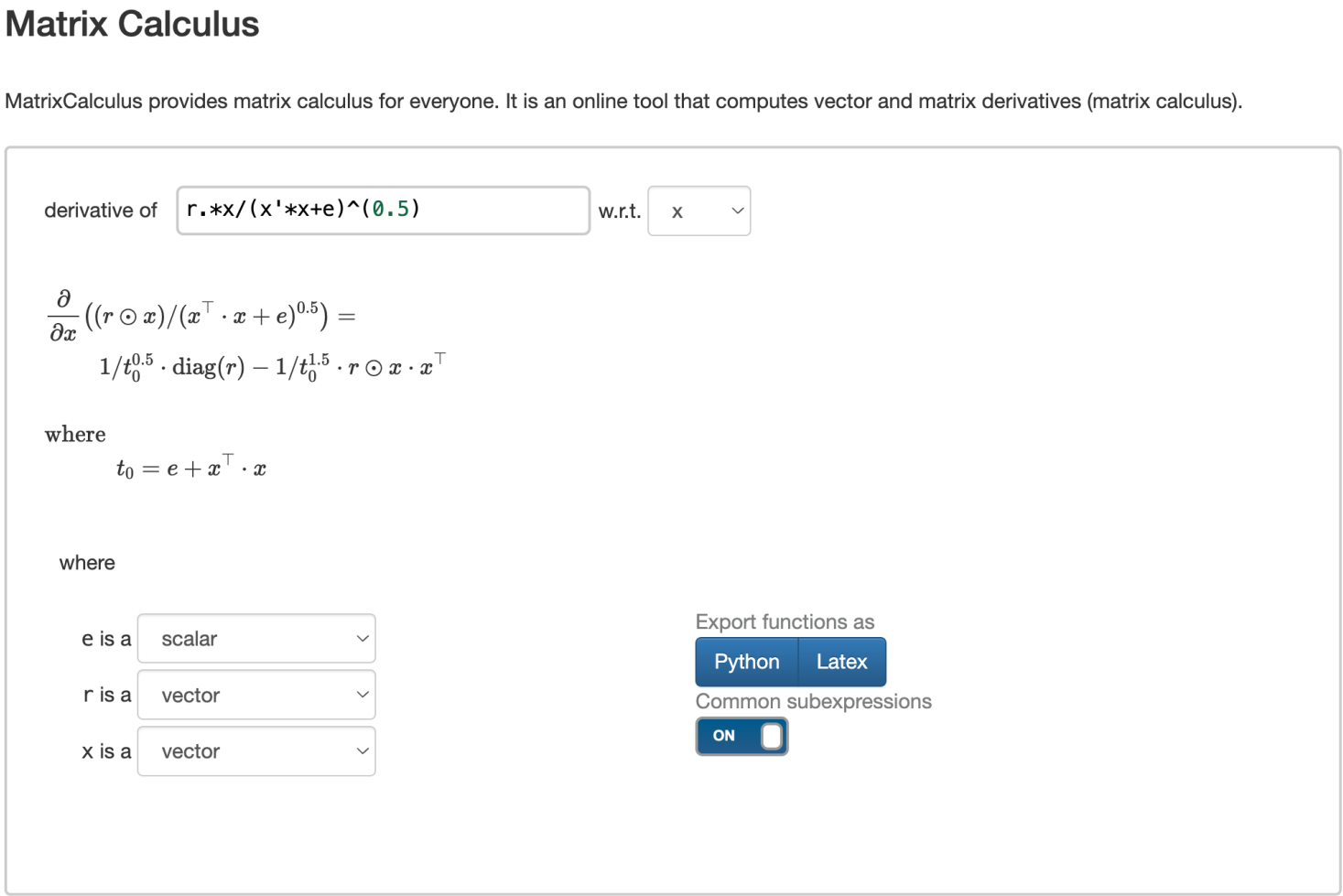



Figure: RMSNorm gradient.

# StableNorm

- Forward Process

$$\text{StableNorm}(\boldsymbol{x}) = \boldsymbol{\gamma} \odot \frac{d^{\alpha}\boldsymbol{x}}{\sqrt{\|\boldsymbol{x}\|_2^2 + \epsilon}}$$

  where $\boldsymbol{x} \in \mathcal{R}^d$

- Backward Process

$$\frac{\partial \,\text{StableNorm}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{d^{\alpha}}{\sqrt{\|\boldsymbol{x}\|_2^2 + \epsilon}} \left( \boldsymbol{I} - \frac{\boldsymbol{x}\boldsymbol{x}^{\top}}{\|\boldsymbol{x}\|_2^2 + \epsilon} \right) \text{Diag}(\boldsymbol{\gamma}).$$

# $L_2$ Norm

- Forward Process

$$\text{L2Norm}(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\sqrt{\|\boldsymbol{x}\|_2^2 + \epsilon}}$$

where $\boldsymbol{x} \in \mathcal{R}^d$

- Backward Process

$$\frac{\partial \, \text{L2Norm}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{1}{\sqrt{\|\boldsymbol{x}\|_2^2 + \epsilon}} \left( \boldsymbol{I} - \frac{\boldsymbol{x}\boldsymbol{x}^\top}{\|\boldsymbol{x}\|_2^2 + \epsilon} \right).$$

# Self-attention with QKNorm

- Forward process of self-attention with QKNorm

$$\boldsymbol{A} = \text{softmax}(\boldsymbol{P}^{(2)}), \quad \text{where } P_{ij}^{(2)} = \left( \frac{\text{RMSN}(\boldsymbol{W}_q \boldsymbol{x}_i)^\top \text{RMSN}(\boldsymbol{W}_k \boldsymbol{x}_j)}{\sqrt{d_1}} \right).$$

- Backward Process

$$\frac{\partial P_{ij}^{(2)}}{\partial \boldsymbol{x}_i} = \frac{\sqrt{d}}{\sqrt{\|\boldsymbol{W}_q \boldsymbol{x}_i\|_2^2 + \epsilon}} \boldsymbol{W}_q{}^\top \left( \boldsymbol{I} - \frac{\boldsymbol{W}_q \boldsymbol{x}_i \left(\boldsymbol{W}_q \boldsymbol{x}_i\right)^\top}{\|\boldsymbol{W}_q \boldsymbol{x}_i\|_2^2 + \epsilon} \right) \text{Diag}(\boldsymbol{\gamma}_q) \, \text{Diag}(\boldsymbol{\gamma}_k) \frac{\boldsymbol{W}_k \boldsymbol{x}_j}{\sqrt{\|\boldsymbol{W}_k \boldsymbol{x}_j\|_2^2 + \epsilon}},$$

$$\frac{\partial P_{ij}^{(2)}}{\partial \boldsymbol{x}_j} = \frac{\sqrt{d}}{\sqrt{\|\boldsymbol{W}_k \boldsymbol{x}_j\|_2^2 + \epsilon}} \boldsymbol{W}_k{}^\top \left( \boldsymbol{I} - \frac{\boldsymbol{W}_k \boldsymbol{x}_j \left(\boldsymbol{W}_k \boldsymbol{x}_j\right)^\top}{\|\boldsymbol{W}_k \boldsymbol{x}_j\|_2^2 + \epsilon} \right) \text{Diag}(\boldsymbol{\gamma}_k) \, \text{Diag}(\boldsymbol{\gamma}_q) \frac{\boldsymbol{W}_q \boldsymbol{x}_i}{\sqrt{\|\boldsymbol{W}_q \boldsymbol{x}_i\|_2^2 + \epsilon}},$$

$$\frac{\partial P_{ij}^{(2)}}{\partial \boldsymbol{W}_q} = \frac{\sqrt{d}}{\sqrt{\|\boldsymbol{W}_q \boldsymbol{x}_i\|_2^2 + \epsilon}} \left( \boldsymbol{I} - \frac{\boldsymbol{W}_q \boldsymbol{x}_i \left(\boldsymbol{W}_q \boldsymbol{x}_i\right)^\top}{\|\boldsymbol{W}_q \boldsymbol{x}_i\|_2^2 + \epsilon} \right) \text{Diag}(\boldsymbol{\gamma}_q) \, \text{Diag}(\boldsymbol{\gamma}_k) \frac{\boldsymbol{W}_k \boldsymbol{x}_j}{\sqrt{\|\boldsymbol{W}_k \boldsymbol{x}_j\|_2^2 + \epsilon}} \boldsymbol{x}_i{}^\top,$$

$$\frac{\partial P_{ij}^{(2)}}{\partial \boldsymbol{W}_k} = \frac{\sqrt{d}}{\sqrt{\|\boldsymbol{W}_k \boldsymbol{x}_j\|_2^2 + \epsilon}} \left( \boldsymbol{I} - \frac{\boldsymbol{W}_k \boldsymbol{x}_j \left(\boldsymbol{W}_k \boldsymbol{x}_j\right)^\top}{\|\boldsymbol{W}_k \boldsymbol{x}_j\|_2^2 + \epsilon} \right) \text{Diag}(\boldsymbol{\gamma}_k) \, \text{Diag}(\boldsymbol{\gamma}_q) \frac{\boldsymbol{W}_q \boldsymbol{x}_i}{\sqrt{\|\boldsymbol{W}_q \boldsymbol{x}_i\|_2^2 + \epsilon}} \boldsymbol{x}_j{}^\top.$$

# Feed-Forward Network with SwishGLU

- Forward Process

$$\boldsymbol{u} = \boldsymbol{W}_1\boldsymbol{x}, \boldsymbol{v} = \boldsymbol{W}_2\boldsymbol{x}, \boldsymbol{z} = \boldsymbol{u} \odot \text{SiLU}(\boldsymbol{v}),$$

$$\boldsymbol{y} = \boldsymbol{W}_3\boldsymbol{z}.$$

  where $\text{SiLU}(\boldsymbol{v}) = \boldsymbol{v} \odot \sigma(\boldsymbol{v})$

- Backward Process

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{W}_3[\text{Diag}(\text{SiLU}(\boldsymbol{W}_2\boldsymbol{x}))\boldsymbol{W}_1 + \text{Diag}(\boldsymbol{W}_1\boldsymbol{x})\text{Diag}(\text{SiLU}'(\boldsymbol{W}_2\boldsymbol{x}))\boldsymbol{W}_2]$$

  where $\text{Diag}(\text{SiLU}'(\boldsymbol{v})) = \text{Diag}\left(\sigma(\boldsymbol{v}) + \boldsymbol{v}\sigma(\boldsymbol{v})\left(1 - \sigma(\boldsymbol{v})\right)\right)$.

# Five Types of Normalizations

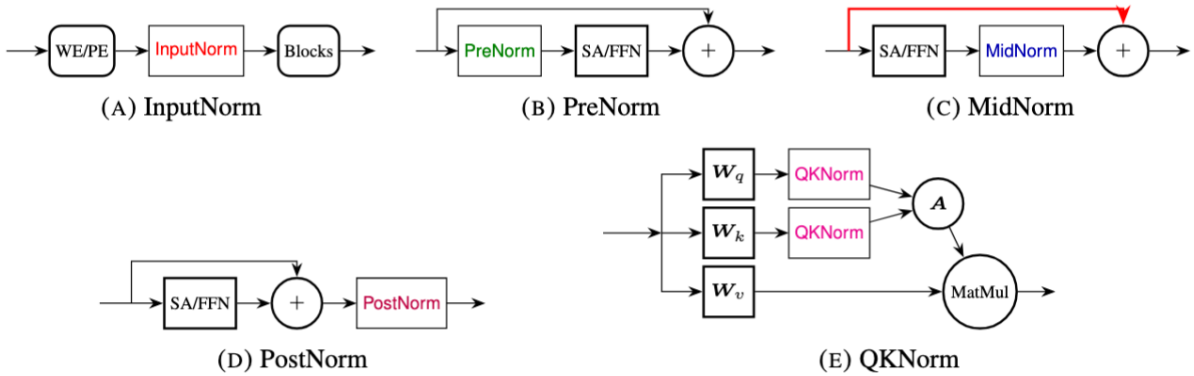



FIGURE 2: Five different normalization methods. The only difference between them is the position of normalization. In (A), WE/PE indicates word embedding and patch embedding.

Figure: Five types of normalizations according to their position in the network.

# Deeply Normalized Transformer (DNT)

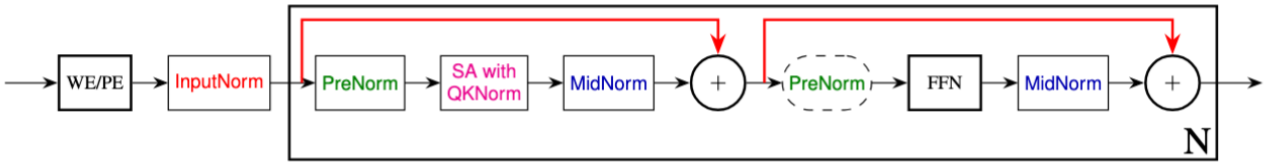
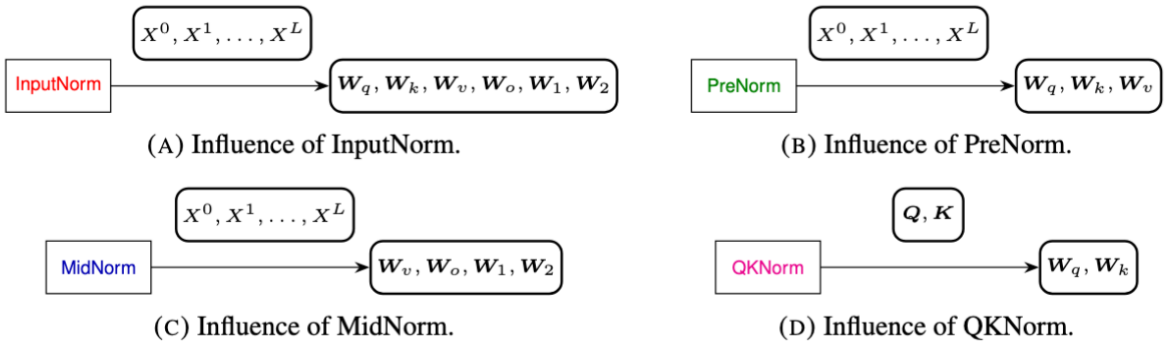



FIGURE 3: DNT architecture. The second PreNorm marked with dashed and rounded corners is optional. By default, we do not use the second PreNorm.

Figure: DNT network.




(A) Influence of InputNorm.

(B) Influence of PreNorm.

(C) Influence of MidNorm.

(D) Influence of QKNorm.

FIGURE 4: Influence of different normalizations. For instance, InputNorm stabilizes $W_q, W_k, W_v, W_o, W_1, W_2$ by constraining $X^0, X^1, \ldots, X^L$.

Figure: Value analysis of each normalization in DNT.

# Unknown problems of mine

We still have many unknown problems

- how to characterize the representation capabilities of transformers? Lipschitz constant? smoothness?
- how to store memory in Transformer?
- how to assess one model without test set?
- what are the characters of a stronger model?
- how can we invent an optimizer that largely improves Adam?
- more and more...

# References I


[1] Sheldon Axler. *Linear algebra done right*. Springer, 2015.

[2] Christopher M Bishop and Hugh Bishop. *Deep learning: Foundations and concepts*. Springer Nature, 2023.

[3] Stanley H Chan. *Introduction to probability for data science*. Michigan Publishing, 2021.

[4] Jeffrey R Chasnov. *Differential Equations for Engineers*. 2019.

[5] Jeffrey R Chasnov. *Matrix Algebra for Engineers*. 2018.

[6] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[7] Alan Edelman and Steven G Johnson. "18. S096 Matrix Calculus for Machine Learning and Beyond, January IAP 2022". In: (2022).

# References II


[8] Lawrence C Evans. *An introduction to stochastic differential equations*. Vol. 82. American Mathematical Soc., 2012.

[9] Alexander Graham. *Kronecker products and matrix calculus with applications*. Courier Dover Publications, 2018.

[10] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[11] Roger A Horn and Charles R Johnson. *Topics in matrix analysis*. Cambridge university press, 1994.

[12] Hyunjik Kim, George Papamakarios, and Andriy Mnih. "The lipschitz constant of self-attention". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5562–5571.

[13] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

# References III


[14] Yurii Nesterov. "Introductory lectures on convex programming volume i: Basic course". In: *Lecture notes* 3.4 (1998), p. 5.

[15] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[16] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. "The matrix cookbook". In: *Technical University of Denmark* 7.15 (2008), p. 510.

[17] Xianbiao Qi, Jianan Wang, and Lei Zhang. "Understanding optimization of deep learning via jacobian matrix and lipschitz constant". In: *arXiv preprint arXiv:2306.09338* (2023).

[18] Xianbiao Qi et al. "Lipsformer: Introducing lipschitz continuity to vision transformers". In: *arXiv preprint arXiv:2304.09856* (2023).

[19] Xianbiao Qi et al. "Stable-Transformer: towards a stable transformer training". In: *arXiv* (2025).

# References IV


[20] Xianbiao Qi et al. "Taming transormer without using learning rate warmup". In: *ICLR* (2025).

[21] Roland Speicher. *High-Dimensional Analysis: Random Matrices and Machine Learning*. Lecture Notes. Department of Mathematics, Saarland University, 2023.

[22] Gilbert Strang. *Linear algebra and learning from data*. SIAM, 2019.

[23] Gilbert Strang, Kenji Hiranabe, and Ashley Fernandes. "The Art of Linear Algebra". In: *PRIMUS* (2024), pp. 1–14.

[24] Lloyd N Trefethen and David Bau. *Numerical linear algebra*. SIAM, 2022.

[25] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

# References V


[26] Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*. Vol. 47. Cambridge university press, 2018.

[27] Ruibin Xiong et al. "On layer normalization in the transformer architecture". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10524–10533.

[28] Raymond W Yeung. *A first course in information theory*. Springer Science & Business Media, 2012.

[29] Biao Zhang and Rico Sennrich. "Root mean square layer normalization". In: *Advances in Neural Information Processing Systems* 32 (2019).

[30] Xianda Zhang. *Matrix analysis and applications*. Cambridge University Press, 2017.