

Lecture 20: Matrix Multiplication & Exponent of Linear Algebra

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

July 8, 2024

Overview

- Matrix Multiplication
- The Exponent of Linear Algebra
- Matrix Inversion
- Determinant and Matrix Inverse
- Computing Partial Derivatives
- Conclusion

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

Compute n matrix vector multiplications.

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

Compute n matrix vector multiplications.

- Running time: $O(n^3)$

Can we do better?

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

Compute n matrix vector multiplications.

- Running time: $O(n^3)$

Can we do better?

- Strassen 1969: YES!
- Idea: divide matrix into blocks, and *reduce number of multiplications* needed!

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \quad P_2 = A_{12}B_{21}, \quad P_3 = S_4B_{22}, \quad P_4 = A_{22}T_4$$

$$P_5 = S_1T_1, \quad P_6 = S_2T_2, \quad P_7 = S_3T_3$$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \quad P_2 = A_{12}B_{21}, \quad P_3 = S_4B_{22}, \quad P_4 = A_{22}T_4$$

$$P_5 = S_1T_1, \quad P_6 = S_2T_2, \quad P_7 = S_3T_3$$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \quad P_2 = A_{12}B_{21}, \quad P_3 = S_4B_{22}, \quad P_4 = A_{22}T_4$$

$$P_5 = S_1T_1, \quad P_6 = S_2T_2, \quad P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \quad P_2 = A_{12}B_{21}, \quad P_3 = S_4B_{22}, \quad P_4 = A_{22}T_4$$

$$P_5 = S_1T_1, \quad P_6 = S_2T_2, \quad P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \quad P_2 = A_{12}B_{21}, \quad P_3 = S_4B_{22}, \quad P_4 = A_{22}T_4$$

$$P_5 = S_1T_1, \quad P_6 = S_2T_2, \quad P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \quad P_2 = A_{12}B_{21}, \quad P_3 = S_4B_{22}, \quad P_4 = A_{22}T_4$$

$$P_5 = S_1T_1, \quad P_6 = S_2T_2, \quad P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22} = P_1 + P_5 + P_6 + P_7$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \quad S_2 = S_1 - A_{11}, \quad S_3 = A_{11} - A_{21}, \quad S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \quad T_2 = B_{22} - T_1, \quad T_3 = B_{22} - B_{12}, \quad T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \quad P_2 = A_{12}B_{21}, \quad P_3 = S_4B_{22}, \quad P_4 = A_{22}T_4$$

$$P_5 = S_1T_1, \quad P_6 = S_2T_2, \quad P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22} = P_1 + P_5 + P_6 + P_7$
- Correctness follows from the computations

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:

- ① 8 additions
- ② 7 multiplications
- ③ 10 additions

S_i, T_i 's
 P_i 's
 C_{ij} 's

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
 - ① 8 additions
 - ② 7 multiplications
 - ③ 10 additions
- Recurrence:

S_i, T_i 's
 P_i 's
 C_{ij} 's

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:

- ① 8 additions
- ② 7 multiplications
- ③ 10 additions

S_i, T_i 's
 P_i 's
 C_{ij} 's

- Recurrence:

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

$$MM(2^k) \leq 7 \cdot MM(2^{k-1}) + 18 \cdot c \cdot 2^{2k-2}$$

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:

- 8 additions
- 7 multiplications
- 10 additions

S_i, T_i 's
 P_i 's
 C_{ij} 's

- Recurrence:

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

$$MM(2^k) \leq 7 \cdot MM(2^{k-1}) + 18 \cdot c \cdot 2^{2k-2}$$

- Could also use Master theorem to get $MM(n) = O(n^{\log 7}) \approx O(n^{2.807})$

Matrix Multiplication Exponent

- We can define ω (or ω_{mult}) as the *matrix multiplication exponent*.
An *algebraic algorithm* for matrix multiplication is an algorithm which only uses the algebraic operations $+$, $-$, \times , \div .
 - 1 If an algebraic algorithm for $n \times n$ matrix multiplication uses $O(n^\alpha)$ operations, then $\omega \leq \alpha$.
 - 2 For any $\varepsilon > 0$, there is an algebraic algorithm for $n \times n$ matrix multiplication which uses $O(n^{\omega+\varepsilon})$ algebraic operations

Matrix Multiplication Exponent

- We can define ω (or ω_{mult}) as the *matrix multiplication exponent*.
An *algebraic algorithm* for matrix multiplication is an algorithm which only uses the algebraic operations $+$, $-$, \times , \div .
 - 1 If an algebraic algorithm for $n \times n$ matrix multiplication uses $O(n^\alpha)$ operations, then $\omega \leq \alpha$.
 - 2 For any $\varepsilon > 0$, there is an algebraic algorithm for $n \times n$ matrix multiplication which uses $O(n^{\omega+\varepsilon})$ algebraic operations
- As we will see today, ω is a fundamental constant in computer science!

Matrix Multiplication Exponent

- We can define ω (or ω_{mult}) as the *matrix multiplication exponent*. An *algebraic algorithm* for matrix multiplication is an algorithm which only uses the algebraic operations $+$, $-$, \times , \div .
 - ① If an algebraic algorithm for $n \times n$ matrix multiplication uses $O(n^\alpha)$ operations, then $\omega \leq \alpha$.
 - ② For any $\varepsilon > 0$, there is an algebraic algorithm for $n \times n$ matrix multiplication which uses $O(n^{\omega+\varepsilon})$ algebraic operations
- As we will see today, ω is a fundamental constant in computer science!
- Currently we know $2 \leq \omega < 2.376$

Open Question

What is the right value of ω ?

Historical Remarks

- Strassen's work is not only important because it gives a faster matrix multiplication algorithm, but because it startled the community that the trivial cubic algorithm could be improved!

Historical Remarks

- Strassen's work is not only important because it gives a faster matrix multiplication algorithm, but because it startled the community that the trivial cubic algorithm could be improved!
- Motivated work on better algorithms for all other linear algebraic problems

Historical Remarks

- Strassen's work is not only important because it gives a faster matrix multiplication algorithm, but because it startled the community that the trivial cubic algorithm could be improved!
- Motivated work on better algorithms for all other linear algebraic problems
- introduced complexity of computation of *bilinear functions* and the study of complexity of tensor decompositions

- Matrix Multiplication
- The Exponent of Linear Algebra
- Matrix Inversion
- Determinant and Matrix Inverse
- Computing Partial Derivatives
- Conclusion

The Exponent of Linear Algebra

- We just saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- How fundamental is the exponent of matrix multiplication?

The Exponent of Linear Algebra

- We just saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- How fundamental is the exponent of matrix multiplication?
- We can similarly define ω_P for a problem P

$\omega_{determinant}$, $\omega_{inverse}$, $\omega_{linear\ system}$, $\omega_{characteristic\ polynomial}$

The Exponent of Linear Algebra

- We just saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- How fundamental is the exponent of matrix multiplication?
- We can similarly define ω_P for a problem P

$\omega_{determinant}$, $\omega_{inverse}$, $\omega_{linear\ system}$, $\omega_{characteristic\ polynomial}$

- As we will see today (and in homework):

$$\omega = \omega_{inverse} = \omega_{determinant}$$

The Exponent of Linear Algebra

- We just saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- How fundamental is the exponent of matrix multiplication?
- We can similarly define ω_P for a problem P

$\omega_{determinant}$, $\omega_{inverse}$, $\omega_{linear\ system}$, $\omega_{characteristic\ polynomial}$

- As we will see today (and in homework):

$$\omega = \omega_{inverse} = \omega_{determinant}$$

- More generally, all of these ω_P 's are related to ω !

Matrix multiplication exponent fundamental to linear algebra!

- Matrix Multiplication
- The Exponent of Linear Algebra
- **Matrix Inversion**
- Determinant and Matrix Inverse
- Computing Partial Derivatives
- Conclusion

Matrix inverse vs matrix multiplication

- Matrix inverse is at least as hard as matrix multiplication
- How to prove this? *reductions!*
 - If we can invert matrices quickly, then we can multiply two matrices quickly.

Matrix inverse vs matrix multiplication

- Matrix inverse is at least as hard as matrix multiplication
- How to prove this? *reductions!*
 - If we can invert matrices quickly, then we can multiply two matrices quickly.
- Suppose we had an algorithm for inverting matrices
- Consider

$$A = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}$$

Matrix inverse vs matrix multiplication

- Matrix inverse is at least as hard as matrix multiplication
- How to prove this? *reductions!*

If we can invert matrices quickly, then we can multiply two matrices quickly.

- Suppose we had an algorithm for inverting matrices
- Consider

$$A = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}$$

- Then

$$A^{-1} = \begin{pmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{pmatrix}$$

Matrix inverse vs matrix multiplication

- Matrix inverse is at least as hard as matrix multiplication
- How to prove this? *reductions!*

If we can invert matrices quickly, then we can multiply two matrices quickly.

- Suppose we had an algorithm for inverting matrices
- Consider

$$A = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}$$

- Then

$$A^{-1} = \begin{pmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{pmatrix}$$

- So if we could invert in time T , then we can multiply two matrices in time $O(T)$.

Matrix Multiplication vs Matrix Inversion

- Matrix multiplication is at least as hard as matrix inversion
“If we can multiply two matrices fast, we can also invert them fast.”

Matrix Multiplication vs Matrix Inversion

- Matrix multiplication is at least as hard as matrix inversion
 - “If we can multiply two matrices fast, we can also invert them fast.”
- Suppose we have an algorithm that performs matrix multiplication.
- Let $n = 2^k$, divide matrix M into blocks of size $n/2$

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

Matrix Multiplication vs Matrix Inversion

- Matrix multiplication is at least as hard as matrix inversion
“If we can multiply two matrices fast, we can also invert them fast.”
- Suppose we have an algorithm that performs matrix multiplication.
- Let $n = 2^k$, divide matrix M into blocks of size $n/2$

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

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible

Matrix Multiplication vs Matrix Inversion

- Matrix multiplication is at least as hard as matrix inversion
“If we can multiply two matrices fast, we can also invert them fast.”
- Suppose we have an algorithm that performs matrix multiplication.
- Let $n = 2^k$, divide matrix M into blocks of size $n/2$

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

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible

- How do we compute this? *Schur Complement*

Similar to how we would invert regular matrices! Just pay attention to non-commutativity.

Runtime Analysis

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible.

Runtime Analysis

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible.

- To invert M , we needed to:
 - Invert A

Runtime Analysis

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible.

- To invert M , we needed to:
 - Invert A
 - Compute $S := D - CA^{-1}B$

Runtime Analysis

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible.

- To invert M , we needed to:
 - Invert A
 - Compute $S := D - CA^{-1}B$
 - Invert S

Runtime Analysis

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible.

- To invert M , we needed to:
 - Invert A
 - Compute $S := D - CA^{-1}B$
 - Invert S
 - perform constant number of multiplications above

Runtime Analysis

- The inverse of M in block form is given by:

$$M^{-1} = \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \cdot \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}$$

Assuming A and $S := D - CA^{-1}B$ are invertible.

- To invert M , we needed to:
 - Invert A
 - Compute $S := D - CA^{-1}B$
 - Invert S
 - perform constant number of multiplications above
- Recurrence relation:

$$I(n) \leq 2 \cdot I(n/2) + C \cdot (n/2)^\omega$$

Solving Recurrence

- Recurrence relation:

$$I(n) \leq 2 \cdot I(n/2) + C \cdot (n/2)^\omega$$

- We know that $2 \leq \omega < 3$

ω is a constant

Solving Recurrence

- Recurrence relation:

$$I(n) \leq 2 \cdot I(n/2) + C \cdot (n/2)^\omega$$

- We know that $2 \leq \omega < 3$
- Recurrence relation:

ω is a constant

$$I(2^k) \leq 2 \cdot I(2^{k-1}) + C \cdot 2^{\omega(k-1)}$$

Solving Recurrence

- Recurrence relation:

$$I(n) \leq 2 \cdot I(n/2) + C \cdot (n/2)^\omega$$

- We know that $2 \leq \omega < 3$

ω is a constant

- Recurrence relation:

$$I(2^k) \leq 2 \cdot I(2^{k-1}) + C \cdot 2^{\omega(k-1)}$$

- Thus

$$\begin{aligned} I(n) = I(2^k) &\leq 2^k \cdot I(1) + C \cdot \sum_{j=0}^{k-1} 2^{\omega j} \\ &\leq C' \cdot \left(2^k + \frac{2^{\omega k} - 1}{2^\omega - 1} \right) \\ &\leq C'' \cdot 2^{\omega k} = C'' n^\omega \end{aligned}$$

Determinant vs Matrix Multiplication

- One can similarly prove that $\omega_{determinant} \leq \omega$
- This is your homework! :)

- Matrix Multiplication
- The Exponent of Linear Algebra
- Matrix Inversion
- **Determinant and Matrix Inverse**
- Computing Partial Derivatives
- Conclusion

Determinant of a Matrix

- Given matrix $M \in \mathbb{F}^{n \times n}$, the determinant is

$$\det(M) = \sum_{\sigma \in S_n} (-1)^\sigma \cdot \prod_{i=1}^n M_{i\sigma(i)}$$

Determinant of a Matrix

- Given matrix $M \in \mathbb{F}^{n \times n}$, the determinant is

$$\det(M) = \sum_{\sigma \in S_n} (-1)^\sigma \cdot \prod_{i=1}^n M_{i\sigma(i)}$$

- Given matrix $M \in \mathbb{F}^{n \times n}$, and $(i, j) \in [n]^2$, the (i, j) -minor of M , denoted $M^{(i,j)}$ is given by

Remove i^{th} row and j^{th} column of M

Determinant of a Matrix

- Given matrix $M \in \mathbb{F}^{n \times n}$, the determinant is

$$\det(M) = \sum_{\sigma \in S_n} (-1)^\sigma \cdot \prod_{i=1}^n M_{i\sigma(i)}$$

- Given matrix $M \in \mathbb{F}^{n \times n}$, and $(i, j) \in [n]^2$, the (i, j) -minor of M , denoted $M^{(i,j)}$ is given by

Remove i^{th} row and j^{th} column of M

- Determinant has a very special decomposition by minors: given any row i , we have

$$\det(M) = \sum_{j=1}^n (-1)^{i+j} M_{i,j} \cdot \det(M^{(i,j)})$$

known as *Laplace Expansion*

Determinant of a Matrix

- Given matrix $M \in \mathbb{F}^{n \times n}$, the determinant is

$$\det(M) = \sum_{\sigma \in S_n} (-1)^\sigma \cdot \prod_{i=1}^n M_{i\sigma(i)}$$

- Given matrix $M \in \mathbb{F}^{n \times n}$, and $(i, j) \in [n]^2$, the (i, j) -minor of M , denoted $M^{(i,j)}$ is given by

Remove i^{th} row and j^{th} column of M

- Determinant has a very special decomposition by minors: given any row i , we have

$$\det(M) = \sum_{j=1}^n (-1)^{i+j} M_{i,j} \cdot \det(M^{(i,j)})$$

known as *Laplace Expansion*

- Determinants of minors are very much related to *derivatives* of the determinant of M

$$\det(M^{(i,j)}) = (-1)^{i+j} \partial_{i,j} \det(M)$$

Determinant and Inverse

- The determinant is intrinsically related to the inverse of a matrix.

Determinant and Inverse

- The determinant is intrinsically related to the inverse of a matrix.
- In particular, let $N \in \mathbb{F}^{n \times n}$ be the *adjugate matrix*

$$N_{i,j} = (-1)^{i+j} \det(M^{(j,i)})$$

Determinant and Inverse

- The determinant is intrinsically related to the inverse of a matrix.
- In particular, let $N \in \mathbb{F}^{n \times n}$ be the *adjugate matrix*

$$N_{i,j} = (-1)^{i+j} \det(M^{(j,i)})$$

- Note that

$$MN = \det(M) \cdot I$$

Determinant and Inverse

- The determinant is intrinsically related to the inverse of a matrix.
- In particular, let $N \in \mathbb{F}^{n \times n}$ be the *adjugate matrix*

$$N_{i,j} = (-1)^{i+j} \det(M^{(j,i)})$$

- Note that

$$MN = \det(M) \cdot I$$

- Entries of the adjugate (determinants of minors) are very much related to *derivatives* of the determinant of M

$$\det(M^{(i,j)}) = (-1)^{i+j} \partial_{i,j} \det(M)$$

Determinant and Inverse

- The determinant is intrinsically related to the inverse of a matrix.
- In particular, let $N \in \mathbb{F}^{n \times n}$ be the *adjugate matrix*

$$N_{i,j} = (-1)^{i+j} \det(M^{(j,i)})$$

- Note that

$$MN = \det(M) \cdot I$$

- Entries of the adjugate (determinants of minors) are very much related to *derivatives* of the determinant of M

$$\det(M^{(i,j)}) = (-1)^{i+j} \partial_{i,j} \det(M)$$

- So, if we knew how to compute the determinant AND ALL its partial derivatives, we could:
 - 1 Compute the adjugate
 - 2 Compute the inverse

Computing the Determinant

- Suppose we have an algorithm which computes the determinant in $O(n^\alpha)$ operations

Computing the Determinant

- Suppose we have an algorithm which computes the determinant in $O(n^\alpha)$ operations
- Can compute the determinant and all its partial derivatives in $O(n^\alpha)$ operations!

Computing the Determinant

- Suppose we have an algorithm which computes the determinant in $O(n^\alpha)$ operations
- Can compute the determinant and all its partial derivatives in $O(n^\alpha)$ operations!
- Compute the inverse by simply dividing $\det(M^{(i,j)}) / \det(M)$

- Matrix Multiplication
- The Exponent of Linear Algebra
- Matrix Inversion
- Determinant and Matrix Inverse
- **Computing Partial Derivatives**
- Conclusion

Algebraic Circuits - base ring R

- Models the *amount of operations* needed to compute polynomial

Algebraic Circuits - base ring R

- Models the *amount of operations* needed to compute polynomial
- *Algebraic Circuit*: directed acyclic graph Φ with
 - input gates labelled by variables x_1, \dots, x_n or elements of R

Algebraic Circuits - base ring R

- Models the *amount of operations* needed to compute polynomial
- *Algebraic Circuit*: directed acyclic graph Φ with
 - input gates labelled by variables x_1, \dots, x_n or elements of R
 - other gates labelled $+$, \times , \div
 - \div gate takes two inputs, which are labelled numerator/denominator

Algebraic Circuits - base ring R

- Models the *amount of operations* needed to compute polynomial
- *Algebraic Circuit*: directed acyclic graph Φ with
 - input gates labelled by variables x_1, \dots, x_n or elements of R
 - other gates labelled $+$, \times , \div
 - \div gate takes two inputs, which are labelled numerator/denominator
 - gates compute polynomial (rational function) in natural way

Algebraic Circuits - base ring R

- Models the *amount of operations* needed to compute polynomial
- *Algebraic Circuit*: directed acyclic graph Φ with
 - input gates labelled by variables x_1, \dots, x_n or elements of R
 - other gates labelled $+$, \times , \div
 - \div gate takes two inputs, which are labelled numerator/denominator
 - gates compute polynomial (rational function) in natural way
- *circuit size*: number of edges in the circuit, denoted by $\mathcal{S}(\Phi)$

Partial Derivatives

- if $f(x_1, \dots, x_n) \in \mathbb{F}[x_1, \dots, x_n]$ the partial derivatives

$$\partial_1 f, \partial_2 f, \dots, \partial_n f$$

are such that

$$\partial_i x_j^d = \begin{cases} dx_j^{d-1}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

and

$$\partial_i f$$

is computed as above considering all other variables “constant”

Partial Derivatives

- if $f(x_1, \dots, x_n) \in \mathbb{F}[x_1, \dots, x_n]$ the partial derivatives

$$\partial_1 f, \partial_2 f, \dots, \partial_n f$$

are such that

$$\partial_i x_j^d = \begin{cases} dx_j^{d-1}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

and

$$\partial_i f$$

is computed as above considering all other variables “constant”

- Example: $f(x_1, x_2) = x_1^2 x_2 - x_1 x_2^3$

$$\partial_1 f = 2x_1 x_2 - x_2^3 \quad \partial_2 f = x_1^2 - 3x_1 x_2^2$$

Partial Derivatives

- if $f(x_1, \dots, x_n) \in \mathbb{F}[x_1, \dots, x_n]$ the partial derivatives

$$\partial_1 f, \partial_2 f, \dots, \partial_n f$$

are such that

$$\partial_i x_j^d = \begin{cases} dx_j^{d-1}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

and

$$\partial_i f$$

is computed as above considering all other variables “constant”

- Example: $f(x_1, x_2) = x_1^2 x_2 - x_1 x_2^3$

$$\partial_1 f = 2x_1 x_2 - x_2^3 \quad \partial_2 f = x_1^2 - 3x_1 x_2^2$$

- How fast can we compute partial derivatives?

Computing Partial Derivatives

- If f can be computed using L operations $+$, $-$, \times , then we can compute *ALL* partial derivatives *simultaneously*

$$\partial_1 f, \dots, \partial_n f$$

performing $4L$ operations!

Computing Partial Derivatives

- If f can be computed using L operations $+$, $-$, \times , then we can compute **ALL** partial derivatives *simultaneously*

$$\partial_1 f, \dots, \partial_n f$$

performing $4L$ operations!

- This is very remarkable, since partial derivatives ubiquitous in computational tasks!
 - 1 gradient descent methods
 - 2 Newton iteration

Computing Partial Derivatives

- If f can be computed using L operations $+$, $-$, \times , then we can compute *ALL* partial derivatives *simultaneously*

$$\partial_1 f, \dots, \partial_n f$$

performing $4L$ operations!

- This is very remarkable, since partial derivatives ubiquitous in computational tasks!
 - ① gradient descent methods
 - ② Newton iteration
- Algorithm we will see today discovered independently in Machine Learning - known as *backpropagation*

Computing Partial Derivatives

- We are going to use the chain rule:

$$\partial_i f(g_1, g_2, \dots, g_m) = \sum_{j=1}^m (\partial_j f)(g_1, g_2, \dots, g_m) \cdot \partial_i g_j$$

Computing Partial Derivatives

- We are going to use the chain rule:

$$\partial_i f(g_1, g_2, \dots, g_m) = \sum_{j=1}^m (\partial_j f)(g_1, g_2, \dots, g_m) \cdot \partial_i g_j$$

- But wait, doesn't the chain rule makes us compute $2m$ partial derivatives?

Computing Partial Derivatives

- We are going to use the chain rule:

$$\partial_i f(g_1, g_2, \dots, g_m) = \sum_{j=1}^m (\partial_j f)(g_1, g_2, \dots, g_m) \cdot \partial_i g_j$$

- But wait, doesn't the chain rule makes us compute $2m$ partial derivatives?
- Main intuitions:
 - 1 if each function we have has m being constant (depend on constant # of variables), then chain rule is **cheap!**

Computing Partial Derivatives

- We are going to use the chain rule:

$$\partial_i f(g_1, g_2, \dots, g_m) = \sum_{j=1}^m (\partial_j f)(g_1, g_2, \dots, g_m) \cdot \partial_i g_j$$

- But wait, doesn't the chain rule makes us compute $2m$ partial derivatives?
- Main intuitions:
 - 1 if each function we have has m being constant (depend on constant # of variables), then chain rule is **cheap**!
 - 2 many of the partial derivatives along the computation will either be zero or *have already been computed*!

Computing Partial Derivatives

- We are going to use the chain rule:

$$\partial_i f(g_1, g_2, \dots, g_m) = \sum_{j=1}^m (\partial_j f)(g_1, g_2, \dots, g_m) \cdot \partial_i g_j$$

- But wait, doesn't the chain rule makes us compute $2m$ partial derivatives?
- Main intuitions:
 - 1 if each function we have has m being constant (depend on constant # of variables), then chain rule is **cheap**!
 - 2 many of the partial derivatives along the computation will either be zero or *have already been computed*!
 - 3 Have to compute partial derivatives “in reverse”

Conclusion

- Today we learned how fundamental matrix multiplication is in symbolic computation and linear algebra
- Used fast computation of partial derivatives to compute the inverse from the determinant