Lecture 20: Matrix Multiplication & Exponent of Linear Algebra

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

July 22, 2021

Overview

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

Rate this course!

Please log in to

https://evaluate.uwaterloo.ca/

- This would really help me figuring out what worked and what didn't for the course
- And let the school know if I was a good boy this term!
- Teaching this course is also a learning experience for me :)

How can I learn more?

Consider taking more advanced courses next term!
See graduate course openings at:

- Current graduate course offerings for next term!

  `https://cs.uwaterloo.ca/current-graduate-students/courses/current-course-offerings/fall-2021-tentative-course-offerings`

- Or, try out some of the research opportunities at UW!

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

Matrix Multiplication

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

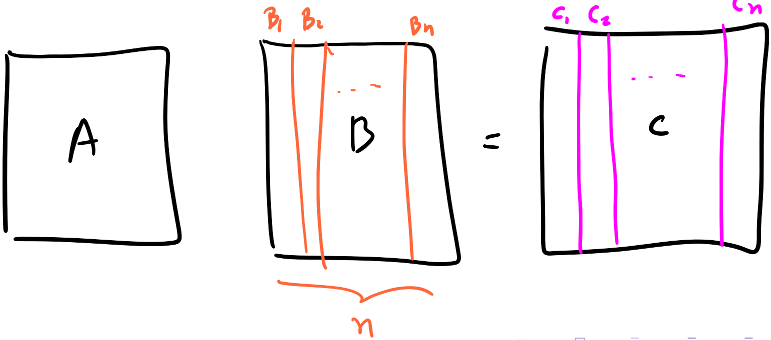
Computational model: number of arithmetic operations (assume that it takes $O(1)$ time to $+, \times, \div, -$)

Matrix Multiplication

$AB_i = C_i$

- **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?

each matrix-vector multiplication takes $O(n^2)$ operations (we need to read all entries of A)

$A_{ij} v_j$

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!

$$\begin{array}{c|c|c|} & n/2 & n/2 \\ \hline n/2 & A_{11} & A_{12} \\ \hline n/2 & A_{21} & A_{22} \\ \hline \end{array}$$

2x2 matrix

if we can compute the product of two 2x2 matrices more efficiently than trivial then we improve running time of Mat. Mul

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 $n/2$:

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

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

$$T(n) \le \underbrace{8}_{\#\text{ of multiplications}} \cdot \underbrace{T(n/2)}_{\text{multiplying } n/2 \times n/2 \text{ matrices}} + \underbrace{c \cdot \left(\frac{n}{2}\right)^2}_{\text{additions}}$$

$\underline{T}(n) \le n^3 = n^{\log_2 8}$

goal: reduce # of multiplications

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:

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

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

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:

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

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

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \; P_2 = A_{12}B_{21}, \; P_3 = S_4 B_{22}, \; P_4 = A_{22} T_4$$

$$P_5 = S_1 T_1, \; P_6 = S_2 T_2, \; P_7 = S_3 T_3$$

Strassen's Algorithm

- Define following matrices:

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

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

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4 B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1 T_1,\ P_6 = S_2 T_2,\ P_7 = S_3 T_3$$

Strassen's Algorithm

- Define following matrices:

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

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

- Compute the following 7 products:

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

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

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

Strassen's Algorithm

- Define following matrices:

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

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

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4 B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1 T_1,\ P_6 = S_2 T_2,\ P_7 = S_3 T_3$$

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

$A_{11}B_{11} + (A_{12} - \cancel{S_1} + \cancel{A_{11}})B_{22} + \cancel{S_1 T_1}$

$+ (\cancel{S_1} - A_{11})(\cancel{B_{22}} - T_1)$

$\cancel{A_{11}B_{11}} + A_{12}B_{22} + \cancel{A_{11}T_1}$ $A_{11}B_{12}$

Strassen's Algorithm

- Define following matrices:

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

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

- Compute the following 7 products:

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

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

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

Strassen's Algorithm

- Define following matrices:

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

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

- Compute the following 7 products:

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

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

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

Strassen's Algorithm

- Define following matrices:

8 +

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

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

- Compute the following 7 products:

7 ×

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

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

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

10 +

- Correctness follows from the computations

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions S_i, T_i's
  2. 7 multiplications P_i's
  3. 10 additions C_{ij}'s

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions $\qquad S_i, T_i$'s
  2. 7 multiplications $\qquad P_i$'s
  3. 10 additions $\qquad C_{ij}$'s
- Recurrence:

$$MM(n) \leq \underbrace{7}_{\text{\# of smaller multiplications}} \cdot MM(n/2) + 18 \cdot \underbrace{c \cdot (n/2)^2}_{\text{time to add 2 } n/2 \times n/2 \text{ matrices}}$$

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions — S_i, T_i's
  2. 7 multiplications — P_i's
  3. 10 additions — C_{ij}'s

$k = \log_2 n$

- Recurrence:

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

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

$$\le 7^2 \cdot MM(2^{k-2}) + 18c \cdot 2^{2k}\left(\frac{1}{4} + \frac{1}{4^2}\right)$$

$$\vdots$$

$$\le 7^k \, MM(1) + 18c\, 2^{2k}\left(\frac{1}{4} + \frac{1}{4^2} + \cdots\right)$$

$$= O(7^{\log_2 n}) = O(n^{\log_2 7})$$

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions S_i, T_i's
  2. 7 multiplications P_i's
  3. 10 additions 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

$2 \leq \omega \leq 2.807 \leq 3$

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

(1) any algorithm that multiplies two $n \times n$ matrices must make at least $c \cdot n^{\omega}$ operations

(lower bound the exponent of any mamu algorithm)

(2) ω is the "best lower bound" on the exponent of any mamu algorithm

Matrix Multiplication Exponent

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

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

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

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

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

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

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

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

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

$\omega_{linear\ systems} \leq \omega$

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

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

- 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!

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

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.

want to prove that $\omega = \omega_{inv}$

we need to prove:

$\omega \geq \omega_{inv}$

$\boxed{\omega \leq \omega_{inv}}$

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

$$M \ \cancel{X} = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix} \quad 3n \times 3n$$

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

$$M \; \cancel{A} = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} I & -A & AB \\ & I & -B \\ & & I \end{pmatrix} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{pmatrix}$$

- Then

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

submatrix of inverse is the multiplication of two n×n matrices!

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

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

$$\boxed{\forall \epsilon > 0 \quad \omega \le \underbrace{\omega_{inv}}_{\alpha} + \epsilon}$$

- 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 $O(T)$.

$O(n^a)$

$O(n^\alpha)$

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."

in ω's words:

$$\boxed{\omega \geq \omega_{inv}}$$

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 $\underbrace{S := D - CA^{-1}B}_{\text{Schur complement of } M}$ 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?

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

Computing Inverse of Block Matrices

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

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

$$\begin{pmatrix} I & B \\ CA^{-1} & D \end{pmatrix}\begin{pmatrix} I & -B \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ CA^{-1} & D - CA^{-1}B \end{pmatrix}$$

$$\begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix}\begin{pmatrix} I & 0 \\ CA^{-1} & D-CA^{-1}B \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & \underbrace{D-CA^{-1}B}_{S} \end{pmatrix}$$

$$\begin{pmatrix} I & 0 \\ 0 & S^{-1} \end{pmatrix}\begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix}\underbrace{\begin{pmatrix} A & B \\ C & D \end{pmatrix}}_{M}\underbrace{\begin{pmatrix} A^{-1} & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} I & -B \\ 0 & I \end{pmatrix}} = I$$

Computing Inverse of Block Matrices

$$\begin{pmatrix} I & -B \\ 0 & I \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & S^{-1} \end{pmatrix}\begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix} M \begin{pmatrix} A^{-1} & 0 \\ 0 & I \end{pmatrix}\underbrace{\begin{pmatrix} I & -B \\ 0 & I \end{pmatrix}\begin{pmatrix} I & B \\ 0 & I \end{pmatrix}}_{I}$$

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

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

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

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.

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

$n \times n$

compute A^{-1}, S^{-1} $\Big\{$ two $n/2 \times n/2$ matrices

$\underbrace{\text{+ some matrix multiplications}}_{}$

we have an algorithm $O(n^{\alpha})$ for this routine!

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:

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

A and S need to be inverted

Solving Recurrence

- Recurrence relation:

$$I(n) \le 2 \cdot I(n/2) + C \cdot (n/2)^{\omega}$$

- We know that $2 \le \omega < 3$ $\qquad$ ω is a constant

Solving Recurrence

- Recurrence relation:

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

- We know that $2 \leq \omega < 3$ $\qquad$ ω is a constant
- Recurrence relation:

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

Solving Recurrence

- Recurrence relation:

$$I(n) \le 2 \cdot I(n/2) + C \cdot (n/2)^{\omega}$$

- We know that $2 \le \omega < 3$ — ω is a constant
- Recurrence relation:

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

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

- Thus

$$I(n) = I(2^k) \le \underline{2^k} \cdot I(1) + C \cdot \sum_{j=0}^{k-1} 2^{\omega j}$$

$$\le C' \cdot \left(\underline{2^k} + \underbrace{\frac{2^{\omega k} - 1}{2^{\omega} - 1}}_{\text{geometric series}}\right)$$

$$\le C'' \cdot 2^{\omega k} = C'' n^{\omega}$$

$$\Rightarrow \boxed{\omega_{inv} \le \omega}$$

Determinant vs Matrix Multiplication

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

Determinant of a Matrix

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

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

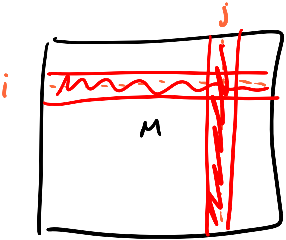
Determinant of a Matrix

- 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

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



$= M^{(i,j)}$

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

$$\det(M) = \sum_{j=1}^{n} (-1)^{i+j} \underbrace{M_{i,j}}_{\text{(i,j) entry of M}} \cdot \overbrace{\det(M^{(i,j)})}^{\text{det. of minor } M^{(i,j)}}$$

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

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

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

derivative w.r.t. variable M_{ij} ← ∂_{ij}

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

known as *Laplace Expansion*

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

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

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

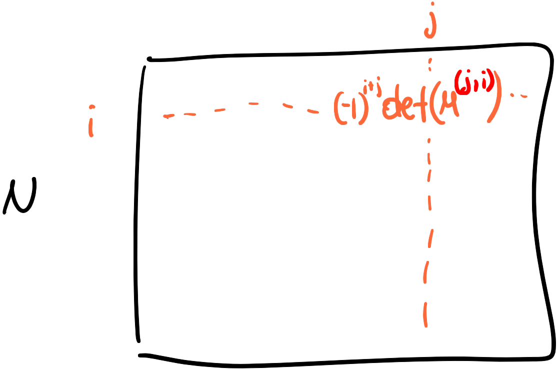
Determinant and Inverse

- 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*

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



note the transposition of indices.

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*

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

- Note that

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

Practice problem: prove the above identity!

$$\underbrace{M \cdot \left(\frac{1}{\det(M)} \cdot N \right)}_{M^{-1}} = I$$

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*

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

- 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

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

$$M^{-1} = \frac{1}{\det(M)} \cdot N \qquad (M^{-1})_{i,j} = \frac{N_{i,j}}{\det(M)} = \frac{(-1)^{i+j} \det(M^{(j,i)})}{\det(M)}$$

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*

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

- 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!

(known in ML as backpropagation)

Computing the Determinant

compute det. in $O(n^\alpha)$ $\Longrightarrow$ compute inverse in $O(n^\alpha)$

equiv: $\omega_{inv} \leq \omega_{det}$ $\quad (\omega \leq \omega_{det})$

- 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 $\boxed{\det(M^{(i,j)})/\det(M)\ (-1)^{i+j}}$

n^2

total runtime: $O(n^\alpha)$ to compute det. and all partial derivatives

$+\ O(n^2)$ to compute all fractions which are the entries of inverse

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

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

Partial Derivatives

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

$$\partial_1 f,\ \partial_2 f, \ldots,\ \partial_n f$$

are such that

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

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

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

  and

  $$\partial_i f$$

  is computed as above considering all other variables "constant"
- Example: $f(x_1, x_2) = x_1^2 x_2 - x_1 x_2^3$

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

Partial Derivatives

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

$$\partial_1 f,\ \partial_2 f, \ldots,\ \partial_n f$$

are such that

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

and

$$\partial_i f$$

is computed as above considering all other variables "constant"

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

$$\partial_1 f = 2x_1x_2 - x_2^3 \qquad \partial_2 f = x_1^2 - 3x_1x_2^2$$

- 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*

$$\partial_1 f, \ldots, \partial_n f$$

performing $4L$ operations!

Computing Partial Derivatives

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

$$\partial_1 f, \ldots, \partial_n f$$

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*

$$\partial_1 f, \ldots, \partial_n f$$

  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:

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

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:

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

- 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:

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

- 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:

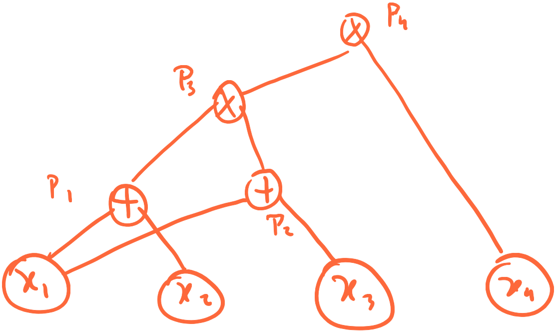
$$\partial_i f(g_1, g_2, \ldots, g_m) = \sum_{j=1}^{m} (\partial_j f)(g_1, g_2, \ldots, g_m) \cdot \partial_i g_j$$

- 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*"

Example

- Consider the following computation:

$$P_1 = x_1 + x_2, \;\; P_2 = x_1 + x_3, \;\; P_3 = P_1 \cdot P_2, \;\; P_4 = x_4 \cdot P_3$$



Example

- Consider the following computation:

$$P_1 = x_1 + x_2,\ P_2 = x_1 + x_3,\ P_3 = P_1 \cdot P_2,\ P_4 = x_4 \cdot P_3$$

- Doing the direct method - i.e. computing all partial derivatives per operation:

Computation	∂_1	∂_2	∂_3	∂_4
$P_1 = x_1 + x_2$	1	1	0	0
$P_2 = x_1 + x_3$	1	0	1	0
$P_3 = P_1 P_2$	$P_2 \cdot \partial_1 P_1 + P_1 \cdot \partial_1 P_2$	$P_2 \cdot \partial_2 P_1$	$P_1 \cdot \partial_3 P_2$	0
$P_4 = x_4 P_3$	$x_4 \cdot \partial_1 P_3$	$x_4 \cdot \partial_2 P_3$	$x_4 \cdot \partial_3 P_3$	P_3

Example

- Consider the following computation:

$$P_1 = x_1 + x_2,\ P_2 = x_1 + x_3,\ P_3 = P_1 \cdot P_2,\ P_4 = x_4 \cdot P_3$$

- Doing the direct method - i.e. computing all partial derivatives per operation:

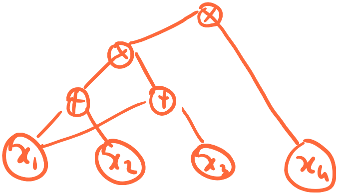
Computation	∂_1	∂_2	∂_3	∂_4
$P_1 = x_1 + x_2$	1	1	0	0
$P_2 = x_1 + x_3$	1	0	1	0
$P_3 = P_1 P_2$	$P_2 \cdot \partial_1 P_1 + P_1 \cdot \partial_1 P_2$	$P_2 \cdot \partial_2 P_1$	$P_1 \cdot \partial_3 P_2$	0
$P_4 = x_4 P_3$	$x_4 \cdot \partial_1 P_3$	$x_4 \cdot \partial_2 P_3$	$x_4 \cdot \partial_3 P_3$	P_3

- Now let's see how to "do it in reverse"

Example - reverse mode

- Consider the computation:

$$P_1 = x_1 + x_2,\ P_2 = x_1 + x_3,\ P_3 = P_1 \cdot P_2,\ P_4 = x_4 \cdot P_3$$



Example - reverse mode

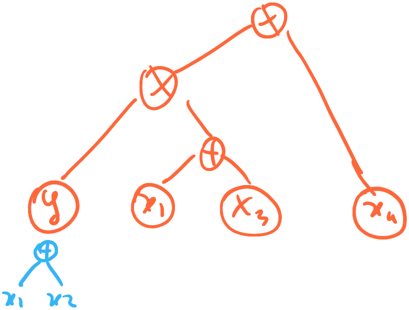
- Consider the computation:

$$\overset{y}{\boxed{P_1 = x_1 + x_2,}}\ P_2 = x_1 + x_3,\ P_3 = P_1 \cdot P_2,\ P_4 = x_4 \cdot P_3$$

- Replacing first computation with a new variable y, we get:

$$Q_2 = x_1 + x_3,\ Q_3 = y \cdot P_2,\ Q_4 = x_4 \cdot P_3$$

Q_4 $\boxed{P_4}$



Example - reverse mode

- Consider the computation:

$$P_1 = x_1 + x_2,\ P_2 = x_1 + x_3,\ P_3 = P_1 \cdot P_2,\ P_4 = x_4 \cdot P_3$$

- Replacing first computation with a new variable y, we get:

$$Q_2 = x_1 + x_3,\ Q_3 = y \cdot P_2,\ Q_4 = x_4 \cdot P_3$$

- Suppose we had an algebraic circuit computing all the partial derivatives of this circuit (including the extra variable y)

Example - reverse mode

- Consider the computation:

$$P_1 = x_1 + x_2,\ P_2 = x_1 + x_3,\ P_3 = P_1 \cdot P_2,\ P_4 = x_4 \cdot P_3$$

- Replacing first computation with a new variable y, we get:

$$Q_2 = x_1 + x_3,\ Q_3 = y \cdot P_2,\ Q_4 = x_4 \cdot P_3$$

- Suppose we had an algebraic circuit computing all the partial derivatives of this circuit (including the extra variable y)
- Can transform the circuit above into one that computes all partial derivatives of P_4 by using the *chain rule*!

Example - reverse mode

- Consider the computation:

$$P_1 = x_1 + x_2,\ P_2 = x_1 + x_3,\ P_3 = P_1 \cdot P_2,\ P_4 = x_4 \cdot P_3$$

- Replacing first computation with a new variable y, we get:

$$Q_2 = x_1 + x_3,\ Q_3 = y \cdot P_2,\ Q_4 = x_4 \cdot P_3$$

- Suppose we had an algebraic circuit computing all the partial derivatives of this circuit (including the extra variable y)
- Can transform the circuit above into one that computes all partial derivatives of P_4 by using the *chain rule*!
- Note that

$$Q_4(x_1, x_2, x_3, x_4, y = P_1) = P_4$$

Computing Partial Derivatives - Proof

- Note that

$$Q_4(x_1, x_2, x_3, x_4, y = P_1) = P_4$$

- By chain rule, we have $\qquad 1 \leq i \leq 4$

$$\begin{aligned}\partial_i Q_4 = &\sum_{j=1}^{4} (\partial_j Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i x_j) \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)\end{aligned}$$

Computing Partial Derivatives - Proof

- Note that

$$Q_4(x_1, x_2, x_3, x_4, y = P_1) = P_4$$

- By chain rule, we have $1 \leq i \leq 4$

$$\begin{aligned}
\partial_i Q_4 = & \sum_{j=1}^{4} (\partial_j Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i x_j) \\
& + (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)
\end{aligned}$$

$$\begin{aligned}
\partial_i Q_4 = & (\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\
& + (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)
\end{aligned}$$

Computing Partial Derivatives - Proof

- Note that
$$Q_4(x_1, x_2, x_3, x_4, y = P_1) = P_4$$
- By chain rule, we have $1 \leq i \leq 4$
$$\begin{aligned}\partial_i Q_4 = &\sum_{j=1}^{4} (\partial_j Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i x_j) \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)\end{aligned}$$

$$\begin{aligned}\partial_i Q_4 = &(\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)\end{aligned}$$

- *Crucial remark*: note that P_1 depends on at most 2 variables!!

Computing Partial Derivatives - Proof

- By chain rule, we have $1 \leq i \leq 4$

$$\begin{aligned}\partial_i Q_4 =& (\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)\end{aligned}$$

Computing Partial Derivatives - Proof

- By chain rule, we have $1 \leq i \leq 4$

$$\begin{aligned} \partial_i Q_4 = & (\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\ & + (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1) \end{aligned}$$

- *Crucial remark*: note that P_1 depends on at most 2 variables!

Computing Partial Derivatives - Proof

- By chain rule, we have $1 \leq i \leq 4$

$$\begin{aligned}\partial_i Q_4 =& (\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)\end{aligned}$$

- *Crucial remark*: note that P_1 depends on at most 2 variables!
- By induction, we know a circuit of size $\leq 4(L-1)$ which computes ALL the $\partial_i Q_4$

Computing Partial Derivatives - Proof

- By chain rule, we have $1 \leq i \leq 4$

$$\begin{aligned}\partial_i Q_4 = &(\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)\end{aligned}$$

- *Crucial remark*: note that P_1 depends on at most 2 variables!
- By induction, we know a circuit of size $\leq 4(L-1)$ which computes ALL the $\partial_i Q_4$
- P_1 is of the form

$$\alpha x_i + \beta x_j, \quad x_i x_j, \quad \alpha x_i + \beta$$

Computing Partial Derivatives - Proof

- By chain rule, we have $1 \leq i \leq 4$

$$\begin{aligned}\partial_i Q_4 =& (\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)\end{aligned}$$

- *Crucial remark*: note that P_1 depends on at most 2 variables!
- By induction, we know a circuit of size $\leq 4(L-1)$ which computes ALL the $\partial_i Q_4$
- P_1 is of the form

$$\alpha x_i + \beta x_j, \quad x_i x_j, \quad \alpha x_i + \beta$$

- So we can compute P_1 and ALL its derivatives with ≤ 4 operations

Computing Partial Derivatives - Proof

- By chain rule, we have $1 \leq i \leq 4$

$$\partial_i Q_4 = (\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot 1 \\ + (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)$$

- *Crucial remark*: note that P_1 depends on at most 2 variables!
- By induction, we know a circuit of size $\leq 4(L-1)$ which computes ALL the $\partial_i Q_4$
- P_1 is of the form

$$\alpha x_i + \beta x_j, \quad x_i x_j, \quad \alpha x_i + \beta$$

- So we can compute P_1 and ALL its derivatives with ≤ 4 operations
- So circuit computing ALL $\partial_i P_4$ derivatives has size

$$\leq 4(L-1) + 4 = 4L$$

Computing Partial Derivatives - Picture