Lecture 20: Partial derivatives & Exponent of Linear Algebra

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

March 29, 2021

Overview

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

The Exponent of Linear Algebra

- Last class we saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- Saw some history on improving on this exponent.

The Exponent of Linear Algebra

- Last class we saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- Saw some history on improving on this exponent.
- Today: how fundamental is the exponent of matrix multiplication?

The Exponent of Linear Algebra

- Last class we saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- Saw some history on improving on this exponent.
- Today: how fundamental is the exponent of matrix multiplication?
- We can similarly define ω_P for a problem P

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

1) $\omega_P \leq \alpha$ if $\exists$ algorithm with $O(n^\alpha)$ operations for problem P

lower bd

2) $\forall \varepsilon > 0$ $\exists$ algorithm with $\alpha < \omega_P + \varepsilon$ with $O(n^\alpha)$ operations.

largest lower bd

The Exponent of Linear Algebra

- Last class we saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- Saw some history on improving on this exponent.
- Today: 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):

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

The Exponent of Linear Algebra

- Last class we saw how to multiply matrices faster than the naive algorithm
- We also learned about $\omega_{mult} := \omega$
- Saw some history on improving on this exponent.
- Today: 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!

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.

invert in $O(T)$ operations $\Longrightarrow$ multiply in $O(T)$ operations

$\Longrightarrow \boxed{\omega \leq \omega_{inversion}}$

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 $n \times n$ $O(n^\alpha)$
- Consider

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

$M \in \mathbb{R}^{3n \times 3n}$

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

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

- Then

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

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.

  $n \times n$ $O(n^\alpha)$

- Suppose we had an algorithm for inverting matrices

  α constant

- Consider

$M \mapsto M^{-1}$

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

$O((3n)^\alpha)$

$= O(3^\alpha \cdot n^\alpha) = O(n^\alpha)$

- Then

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

$\therefore$ multiplication takes $O(n^\alpha)$

- 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."

$\Rightarrow \quad \omega_{inverse} \leq \omega$

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.

  in $O(n^\alpha)$ operations
- Let $n = 2^k$, divide matrix M into blocks of size $n/2$

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

$2^k \le n < 2^{k+1}$

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

To invert M we need: compute $\left.\begin{matrix} A^{-1} \\ S^{-1} \end{matrix}\right\}$ two inversions of size $\frac{n}{2} \times \frac{n}{2}$

+ constantly many $\frac{n}{2} \times \frac{n}{2}$ matrix multiplications

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

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

$$\underbrace{\begin{pmatrix} I & A^{-1}B \\ 0 & S \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & S^{-1} \end{pmatrix}}_{\begin{pmatrix} I & A^{-1}BS^{-1} \\ 0 & I \end{pmatrix}} \begin{pmatrix} I & -A^{-1}BS^{-1} \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} = I_n$$

Computing Inverse of Block Matrices

$$M \cdot M^{-1} = I$$

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

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

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

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

$$I = \begin{pmatrix} I & 0 \\ 0 & S^{-1} \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & S \end{pmatrix}$$

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 $S := D - CA^{-1}B$

2 matrix multiplications

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

$A^{-1}BS^{-1}$ CA^{-1}

- 8 matrix multiplications for 2x2 block matrix multiplication.

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:

$$I(n) \leq 2 \cdot I(n/2) + \underbrace{C}_{\#\text{ matrix multiplications}} \cdot \overbrace{(n/2)^{\omega}}^{\text{matrix multiplication time}}$$

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 $2 \leq \omega < 3$ $\qquad$ 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:

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

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

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

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

$$= 2^2 \cdot I(2^{k-2}) + C\left(2^{\omega(k-1)} + \underbrace{2 \cdot 2^{\omega(k-2)}}_{< \frac{2^{\omega(k-1)}}{2}}\right)$$

- Thus

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

$$\leq C' \cdot \left(2^k + \frac{2^{\omega k} - 1}{2^{\omega} - 1}\right)$$

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

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

Determinant vs Matrix Multiplication

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

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 $x_1, \ldots, x_n$ or elements of R

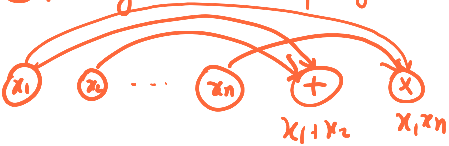
Algebraic Circuits - base ring R

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

Algebraic Circuits - base ring R

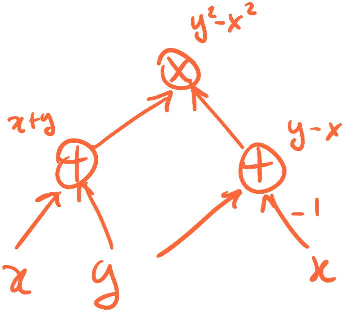
- Models the *amount of operations* needed to compute polynomial
- *Algebraic Circuit*: directed acyclic graph Φ with
  - input gates labelled by variables $x_1, \ldots, x_n$ or elements of R
  - other gates labelled $+, \times, \div$
  - $\div$ gate takes two inputs, which are labelled numerator/denominator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

Straight-line programs



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 $x_1, \ldots, x_n$ 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 $f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ 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: $f(x_1, x_2) = x_1^2 x_2 - x_1 x_2^3$

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

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

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

- 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*

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

performing $4L$ operations!

Naive algorithm : $O(L \cdot n)$

Today : $O(L)$

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

$g_j(x_1, \ldots, x_n)$

- We are going to use the chain rule:

$$\partial_i f(g_1, g_2, \ldots, g_m) = \sum_{j=1}^{m} \underbrace{(\partial_j f)}_{y \text{ variables}}(g_1, g_2, \ldots, g_m) \cdot \underbrace{\partial_i g_j}_{x_i \text{ variable}}$$

$f(y_1, \ldots, y_m)$

$\partial_j f(y_1, \ldots, y_m)\Big|_{y_i = g_i}$

Computing Partial Derivatives

- We are going to use the chain rule:

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

- 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**!

$m = 2 \Rightarrow 2m = 4$ 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**!
  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:

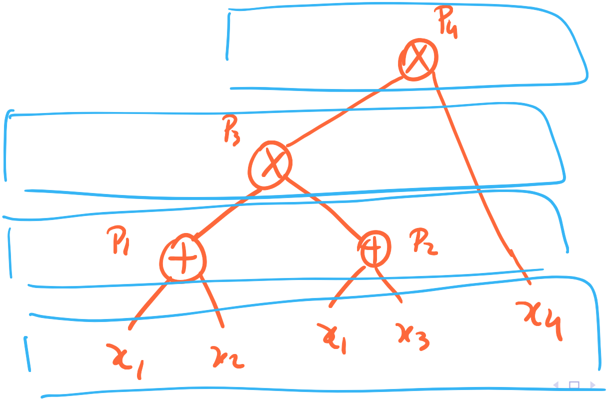
$$\partial_i f(g_1, g_2, \ldots, g_m) = \sum_{j=1}^{m} (\partial_j f)(g_1, g_2, \ldots, g_m) \cdot \partial_i g_j$$

- 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: x_1, x_2, x_3, x_4

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



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:

n variables

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

L operations

$O(L \cdot n)$

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:

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

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

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

n+1 variables

$\boxed{L-1 \text{ operations !}}$

Idea: compute in reverse by induction on circuit size (# operations)

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)

$4(L-1)$ operations (by induction)

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

$\partial_i P_4$ want to compute

- 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

$$Q_4(x_1, x_2, x_3, x_4, \boxed{y = P_1}) = P_4$$

$\partial_i Q_4(x_1, \ldots, x_n, y)$ $\quad$ $\partial_y Q_4(x_1, \ldots, x_n, y)$ $\quad$ in $4(\ell-1)$ ops.

Computing Partial Derivatives - Proof

$P_1 = x_1 + x_2$

- Note that

$$Q_4(x_1, x_2, x_3, x_4, \underline{y = P_1}) = P_4$$

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

$$\begin{aligned}\partial_i P_4 = \sum_{j=1}^{4} &(\partial_j Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot \boxed{(\partial_i x_j)} \\ &+ (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1)\, \boxed{(\partial_i P_1)}\end{aligned}$$

$$\boxed{Q_4(x_1, x_2, x_3, x_4, P_1) = P_4}$$

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 $\quad 1 \le i \le 4$

$$\partial_i P_4 = \sum_{j=1}^{4} (\partial_j Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i x_j) + (\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1) \cdot (\partial_i P_1)$$

constant!

$$\partial_i P_4 \quad \cancel{\partial_i Q_4} = \underline{(\partial_i Q_4)(x_1, x_2, x_3, x_4, P_1)} \cdot 1 + \underline{(\partial_y Q_4)(x_1, x_2, x_3, x_4, P_1)} \cdot \boxed{(\partial_i P_1)}$$

$P_1 = x_1 + x_2$

computed by induction!

1 if $i \in \{1, 2\}$

0 otherwise

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 $\qquad 1 \leq i \leq 4$

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

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

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

Computing Partial Derivatives - Proof

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

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

precomputed

$$P_1 = \begin{cases} x_1 + x_2 \\ x_1 - x_2 \\ x_1 x_2 \\ x_1 + \alpha \end{cases}$$

$$\boxed{\pm 1,\ 0,\ x_1,\ x_2}$$

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 $\partial_i Q_4$, $\partial_y Q_4$

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 $1 \leq i \leq 4$

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

- *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

$$\boxed{\alpha x_i + \beta x_j,} \quad x_i x_j, \quad \alpha x_i + \beta$$

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

Computing Partial Derivatives - Proof

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

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

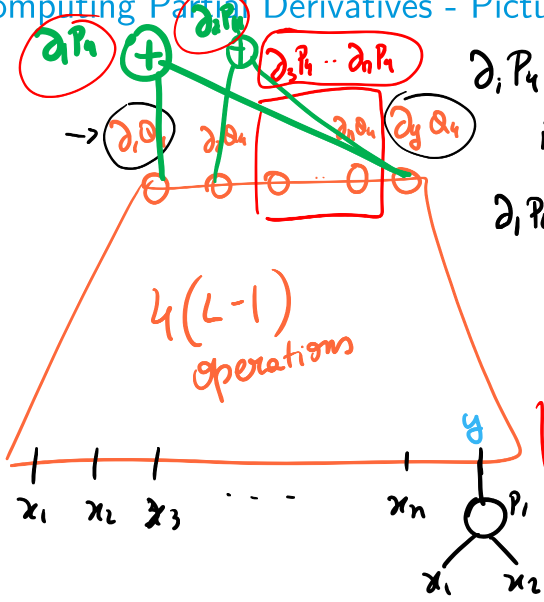
- *Crucial remark*: note that P_1 depends on at most 2 variables!
- By induction, we know a circuit of size $\leq 4(L-1)$ which computes ALL the $\partial_i Q_4$
- P_1 is of the form

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

- 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



$$\partial_i P_4 = \partial_i Q_4(\bar{x}, P_1)\cdot 1$$

$$i \neq 1, 2$$

$$\partial_1 P_4 = \partial_1 Q_4(\bar{x}, P_1)\cdot 1 + \partial_y Q_4(\bar{x}, P_1)\cdot \underbrace{\partial_1 P_1}_{1}$$

substitution of $y = P_1$

Determinant of a Matrix

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

$$\det(M) = \sum_{\sigma \in S_n} \underbrace{(-1)^{\sigma}}_{\text{sign of permutation}} \cdot \prod_{i=1}^{n} M_{i\sigma(i)}$$

$n!$ operations

Gaussian elimination: $O(n^3)$ operations

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

(1,2) minor

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \longrightarrow M^{(1,2)} = \begin{pmatrix} 4 & 6 \\ 7 & 9 \end{pmatrix}$$

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
$$\det(M) = \sum_{j=1}^{n} (-1)^{i+j} M_{i,j} \cdot \det(M^{(i,j)})$$
known as *Laplace Expansion* columns

Determinant of a Matrix

$M = (M_{ij})$ ← variable

- 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

$M \quad i \left(\cancel{\quad M_{ij} \quad} \right)$

$$\det(M) = \sum_{j=1}^{n} (-1)^{i+j} M_{i,j} \cdot \overbrace{\det(M^{(i,j)})}^{\text{don't depend } M_{i,*}}$$

doesn't depend on $M^{(i,j)}$

known as *Laplace Expansion*

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

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

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*

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

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

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*

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

- Note that

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

$$\boxed{\frac{1}{\det(M)} N = M^{-1}}$$

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

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

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!

by our derivatives computation
(back propagation)

Computing the Determinant

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

$\Rightarrow$ $\boxed{\omega_{determinant} \geq \omega}$

- 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 $\underbrace{\det(M^{(i,j)})}_{\text{constant multiple of partial derivatives}} / \det(M)$

Conclusion

- Today we learned how fundamental matrix multiplication is in symbolic computation and linear algebra
- Learned how to compute ALL partial derivatives efficiently - roughly the same time it takes to compute our polynomial!
- Used fast computation of partial derivative to compute the determinant

Acknowledgement

- Lecture based largely on:
  - Eric Schost's notes
  - Survey by Shpilka and Yehudayoff

    https://www.nowpublishers.com/article/Details/TCS-039