CS 4803 / 7643: Deep Learning

Topics:

- Automatic Differentiation
  - (Finish) Forward mode vs Reverse mode AD
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs
Administrativia

• HW1 Reminder
  – Due: 09/26, 11:55pm
Project

• Goal
  – Chance to take on something open-ended
  – Encouraged to apply to your research
    (computer vision, NLP, robotics,…)

• Main categories
  - Application/Survey
    - Compare a collection of existing algorithms on a new application
domain of your interest
  - Formulation/Development
    - Formulate a new model or algorithm for a new or old problem
  - Theory
    - Theoretically analyze an existing algorithm
Project

• Rules
  – Combine with other classes / research / credits / anything
    • You have our blanket permission
    • Get permission from other instructors; delineate different parts
  – Must be done this semester.
  – Groups of 3-4

• Expectations
  – 20% of final grade = individual effort equivalent to 1 HW
  – Expectation scales with team size
  – Most work will be done in Nov but please plan early.
Project Ideas

- NeurIPS Reproducibility Challenge
Computing

• Major bottleneck
  – GPUs

• Options
  – Your own / group / advisor’s resources

  ⬤ Google Cloud Credits
    • $50 credits to every registered student courtesy Google

  – Google Colab
    • jupyter-notbook + free GPU instance
Administrativia

• Project Teams Google Doc
  – https://docs.google.com/spreadsheets/d/1ouD6ctaemV_3nb2MQHs7rUOAaW9DFLu8I5Zd3yOFs7E/edit?usp=sharing
  – Project Title
  – 1-3 sentence project summary TL;DR
  – Team member names
Recap from last time
How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”
Chain Rule: Composite Functions

\[ h(x) = f(g(x)) = (f \circ g)(x) \]

\[
\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx} = (g \circ f \circ g^{-1} \cdots g_1^{-1})(x)
\]
Chain Rule: Scalar Case

\[ x \xrightarrow{g(.)} z \xrightarrow{f(.)} y \rightarrow a \quad x, y, z \in \mathbb{R}, a \in \mathbb{R} \]

\[ f(g(x)) = f(z) \]

\[ \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} \]

Scalar prod.

\[ \frac{\partial a}{\partial x} = \frac{\partial a}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} \]
Chain Rule: Vector Case

\[ \overrightarrow{X} \in \mathbb{R}^d \xrightarrow{g} \overrightarrow{Z} \in \mathbb{R}^m \xrightarrow{f} \overrightarrow{y} \in \mathbb{R}^c \rightarrow \overrightarrow{a} \]

\[ \frac{\partial \overrightarrow{y}}{\partial \overrightarrow{x}} \xrightarrow{J} \frac{\partial \overrightarrow{y}}{\partial \overrightarrow{z}} \xrightarrow{0} \frac{\partial \overrightarrow{z}}{\partial \overrightarrow{x}} \]

\[ J_{f \circ g} \quad J_f \quad \text{Matrix Mult} \]
Chain Rule: Jacobian view

\[ \frac{\partial y_i}{\partial x_j} = \sum_k \frac{\partial^2 y_i}{\partial z_k \partial x_j} \]

\[ \begin{bmatrix} \frac{\partial y_i}{\partial x_j} \\ \vdots \end{bmatrix} _{c \times d} = \begin{bmatrix} \frac{\partial y_i}{\partial z_k} \\ \vdots \end{bmatrix} _{c \times m} \begin{bmatrix} \frac{\partial z_k}{\partial x_j} \\ \vdots \end{bmatrix} _{m \times d} \]
Chain Rule: Graphical view

\[
\frac{\partial y_i}{\partial x_j} = \sum_{k \text{ is on path}} \frac{\partial y_i}{\partial z_k} \cdot \frac{\partial z_k}{\partial x_j}
\]
Chain Rule: Cascaded

\[ \frac{\partial h}{\partial h^0} = -\frac{\partial L}{\partial h^0} \cdot \frac{\partial h^0}{\partial h^{l-1}} \cdot \frac{\partial h^{l-1}}{\partial h^{l-2}} \cdot \cdots \cdot \frac{\partial h^1}{\partial h^0} \]

\[ \Theta(d^3) \]

\[ O(d^3) \]
Deep Learning = Differentiable Programming

• Computation = Graph
  – Input = Data + Parameters
  – Output = Loss
  – Scheduling = Topological ordering

• Auto-Diff
  – A family of algorithms for implementing chain-rule on computation graphs
Directed Acyclic Graphs (DAGs)

- Exactly what the name suggests
  - Directed edges
  - No (directed) cycles
  - Underlying undirected cycles okay

\[ G = (V, E) \]
\[ E = \{ (v_i, v_j) \mid v_i, v_j \in V \} \]
Directed Acyclic Graphs (DAGs)

- Concept
  - Topological Ordering

\[ \exists \text{ bijection } \sigma : V \rightarrow \{1, \ldots, n\} \]
\[ \text{s.t. } \forall (v_i, v_j) \in E \]
\[ \sigma(v_i) < \sigma(v_j) \]
Computational Graphs

- Notation

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]
Deep Learning = Differentiable Programming

• Computation = Graph
  – Input = Data + Parameters
  – Output = Loss
  – Scheduling = Topological ordering

• Auto-Diff
  – A family of algorithms for implementing chain-rule on computation graphs
Forward mode AD

Goal: \( \frac{\partial L}{\partial x} \)

Layer \( l \)

Input:

\( \frac{\partial h^{l}}{\partial x} \)

\( h^{l-1} \)

\( h = g(h^{l-1}) \)

\( \frac{\partial h^{l}}{\partial x} = \frac{\partial h^{l}}{\partial h^{l-1}} \cdot \frac{\partial h^{l-1}}{\partial x} \)

Jacobian of \( g \)
Reverse mode AD

\[ \frac{\partial y}{\partial x} = A \]
\[ \frac{\partial x}{\partial y} = A^{-1} \]
\[ \frac{\partial h}{\partial x} = W \]
\[ \frac{\partial h}{\partial x} = W \]
\[ \frac{\partial h}{\partial x} = W \]

Goal: \[ \frac{\partial L}{\partial x} \]

\[ \frac{\partial L}{\partial h} = g(h^{l-1}) \]

Output

\[ \frac{\partial L}{\partial h^{l-1}} \]

Input

\[ \frac{\partial L}{\partial h} \]

Jacobian of g
Plan for Today

• Automatic Differentiation
  – (Finish) Forward mode vs Reverse mode AD
  – Backprop
  – Patterns in backprop
  – Jacobians in FC+ReLU NNs
Example: Forward mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]
Example: Forward mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]
Example: Forward mode AD

\[
f(x_1, x_2) = x_1 x_2 + \sin(x_1)
\]
Example: Forward mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]

\[ \dot{w}_3 = \dot{w}_1 + \dot{w}_2 \]

\[ \dot{w}_1 = \cos(x_1) \dot{x}_1 \]
\[ \dot{w}_2 = \dot{x}_1 x_2 + x_1 \dot{x}_2 \]

\[ x_1 = \frac{\partial f}{\partial x_1} \]
\[ \dot{x}_1 = \frac{\partial f}{\partial x_1} \]
\[ \dot{x}_2 = \frac{\partial f}{\partial x_2} \]
Example: Forward mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) + \cos(x_3) \]

Q: What happens if there’s another input variable \( x_3 \)?
Example: Forward mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]

Q: What happens if there’s another input variable \( x_3 \)?

A: more sophisticated graph; d “forward props” for d variables
Example: Forward mode AD

\[ f_1(x_1, x_2) = x_1 x_2 + \sin(x_1) \quad f_2 = \cos(x_2) \]

Q: What happens if there’s another output variable \( f_2 \)?
Example: Forward mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]

Q: What happens if there’s another output variable \( f_2 \)?
A: more sophisticated graph; single “forward prop”
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]
Gradients add at branches

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) + \cos(x_3) \]

Q: What happens if there’s another input variable \( x_3 \)?
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]

Q: What happens if there’s another input variable \( x_3 \)?
A: more sophisticated graph; single “backward prop”
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]

Q: What happens if there’s another output variable \( f_2 \)?

For on output vars:

\[ \bar{x}_1 = \bar{w}_1 \cos(x_1) \quad \bar{x}_1 = \bar{w}_2 x_2 \quad \bar{x}_2 = \bar{w}_2 x_1 \]

\[ \bar{w}_1 = \bar{w}_3 \quad \bar{w}_2 = \bar{w}_3 \quad \bar{w}_3 = 1 \]
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]

Q: What happens if there’s another output variable \( f_2 \)?
A: more sophisticated graph; c “backward props” for c vars

\[
\begin{align*}
\bar{w}_3 &= 1 \\
\bar{w}_1 &= \bar{w}_3 \\
\bar{w}_2 &= \bar{w}_3 \\
\bar{x}_1 &= \bar{w}_1 \cos(x_1) \\
\bar{x}_1 &= \bar{w}_2 x_2 \\
\bar{x}_2 &= \bar{w}_2 x_1
\end{align*}
\]
Forward mode vs Reverse Mode

- $x \rightarrow \text{Graph} \rightarrow L$
- Intuition of Jacobian
Forward mode vs Reverse Mode

• What are the differences?

• Which one is faster to compute?
  – Forward or backward?

Is $c > d$ or $c < d$?
Forward mode vs Reverse Mode

• What are the differences?

• Which one is faster to compute?
  – Forward or backward?

• Which one is more memory efficient (less storage)?
  – Forward or backward?
Forward Pass vs Forward mode AD vs Reverse Mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]
Plan for Today

• Automatic Differentiation
  – (Finish) Forward mode vs Reverse mode AD ✓
  – Backprop
  – Patterns in backprop
  – Jacobians in FC+ReLU NNs
Any DAG of differentiable modules is allowed!
Key Computation: Forward-Prop
Key Computation: Back-Prop
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
Neural Network Training

- Step 1: Compute Loss on mini-batch

[Slide Credit: Marc'Aurelio Ranzato, Yann LeCun]
Neural Network Training

- Step 1: Compute Loss on mini-batch

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]
Neural Network Training

- Step 1: Compute Loss on mini-batch [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]
Neural Network Training

• Step 1: Compute Loss on mini-batch [F-Pass]
• Step 2: Compute gradients wrt parameters [B-Pass]
Neural Network Training

• Step 1: Compute Loss on mini-batch [F-Pass]
• Step 2: Compute gradients wrt parameters [B-Pass]
• Step 3: Use gradient to update parameters

\[ \theta \leftarrow \theta - \eta \frac{dL}{d\theta} \]