CS 4803 / 7643: Deep Learning

Topics:

- Automatic Differentiation
  - Patterns in backprop
  - Jacobians in FC+ReLU NNs

Dhruv Batra
Georgia Tech
HW1 Reminder
- Due: 09/26, 11:55pm

Project Teams Google Doc
- [https://docs.google.com/spreadsheets/d/1ouD6ctaemV_3nb2MQHs7rUOAaW9DFLu8l5Zd3yOFs7E/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1ouD6ctaemV_3nb2MQHs7rUOAaW9DFLu8l5Zd3yOFs7E/edit?usp=sharing)
  - Project Title
  - 1-3 sentence project summary TL;DR
  - Team member names
Recap from last time
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

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

layer \( l \)

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

\[ \hat{h} \]

\[ \hat{h}^{l-1} \]

\[ \frac{\partial h^l}{\partial x} = \frac{\partial h^l}{\partial h^{l-1}} \cdot \frac{\partial h^{l-1}}{\partial x} \]

Jacobian of \( g \)

Input:

\[ \left[ \frac{\partial h^l}{\partial x} \right] \]
Reverse mode AD

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

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

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

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

Goal: \[ \frac{\partial L}{\partial x} \]
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: Reverse mode AD

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

\[ \bar{w}_3 = 1 \]

\[ \bar{w}_1 = \bar{w}_3 \quad \bar{w}_2 = \bar{w}_3 \]

\[ \bar{x}_1 = \bar{w}_1 \cos(x_1) \quad \bar{x}_1 = \bar{w}_2 x_2 \quad \bar{x}_2 = \bar{w}_2 x_1 \]
Forward mode vs Reverse Mode

- **x → Graph → L**
- **Intuition of Jacobian**
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?
Any DAG of differentiable modules is allowed!
Key Computation: Forward-Prop

\[ X \rightarrow \theta \rightarrow Z \]
Key Computation: Back-Prop

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

\[ \left\{ \frac{\partial Z}{\partial X}, \frac{\partial Z}{\partial \theta} \right\} \]

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

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

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

• Automatic Differentiation
  – Patterns in backprop
  – Jacobians in FC+ReLU NNs
Neural Network Computation Graph

\[ \mathbf{x} \rightarrow f_1 \rightarrow h^{(1)} \rightarrow f_2 \rightarrow h^{(2)} \rightarrow \cdots \rightarrow f_L \rightarrow L \in \mathbb{R} \]

\[ \frac{\partial L}{\partial \mathbf{w}_1}, \frac{\partial L}{\partial \mathbf{w}_2}, \ldots, \frac{\partial L}{\partial \mathbf{w}_L} \]
Backprop
Modularized implementation: forward / backward API

Graph (or Net) object  (rough pseudo code)

```python
class ComputationalGraph(object):
    # ...
    def forward(inputs):
        # 1. pass inputs to input gates...
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss

def backward():
    for gate in reversed(self.graph.nodes_topologically_sorted()):
        gate.backward() # little piece of backprop (chain rule applied)
    return inputs_gradients
```
Modularized implementation: forward / backward API

(x, y, z are scalars)
Modularized implementation: forward / backward API

\[(x, y, z \text{ are scalars})\]

```python
class MultiplyGate(object):
    def forward(self, x, y):
        z = x * y
        self.x = x  # must keep these around!
        self.y = y
        return z

    def backward(self, dz):
        dx = self.y * dz  # [dz/dx * dL/dz]
        dy = self.x * dz  # [dz/dy * dL/dz]
        return [dx, dy]
```

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Example: Caffe layers

Caffe is licensed under BSD 2-Clause.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Caffe Sigmoid Layer

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

* top_diff (chain rule)
Plan for Today

• Automatic Differentiation
  – Patterns in backprop
  – Jacobians in FC+ReLU NNs
Backpropagation: a simple example

\[ f(x, y, z, n) = 2 \left( xy + \max \left\{ z, w \right\} \right) \]

Diagram:

- \( x = 3.00 \)
- \( y = -4.00 \)
- \( z = 2.00 \)
- \( w = -1.00 \)
- \( n \)

\( \max \) operation selects the maximum between \( z \) and \( w \).

\( f = \frac{\partial f}{\partial f} = 1 \)

\( \frac{\partial f}{\partial f} = 2 \)
Backpropagation: a simple example
Patterns in backward flow
Patterns in backward flow

Q: What is an add gate?

\[ w_3 = w_1 + w_2 \]

\[ \frac{\partial F}{\partial w_1} = \frac{\partial F}{\partial w_3} \cdot \frac{\partial w_3}{\partial w_1} \]

\[ = w_3 \begin{bmatrix} 1 \\ -1 \end{bmatrix} \]

\[ \overline{w}_2 = \overline{w}_3 \]
Patterns in backward flow

**add** gate: gradient distributor
Patterns in backward flow

**Q:** What is a **max** gate?

- **add** gate: gradient distributor

\[
\begin{align*}
    w_2 &= \begin{cases} 
    z & \text{if } z > w \\
    w & \text{else}
    \end{cases} \\
    \bar{z} &= \frac{\partial F}{\partial w_2} = \frac{\partial F}{\partial z} \cdot \frac{\partial w_2}{\partial z} \\
    \bar{z} &= \begin{cases} 
    +1 & \text{if } z > w \\
    0 & \text{else}
    \end{cases}
\end{align*}
\]

\[w_2 = \max\{z, w\} \quad \text{max}\{0, x\}\]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Patterns in backward flow

**add** gate: gradient distributor

**max** gate: gradient router

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
add gate: gradient distributor
max gate: gradient router
Q: What is a **mul** gate?

Patterns in backward flow

---

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Patterns in backward flow

**Add** gate: gradient distributor

**Max** gate: gradient router

**Mul** gate: gradient switcher

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Gradients add at branches
Duality in Fprop and Bprop

Diagrams:

- **SUM**:
  - **FPROP**: Red arrow entering a sum symbol, followed by a black arrow.
  - **BPROP**: Black arrow leaving a sum symbol, followed by a red arrow.

- **COPY**:
  - **FPROP**: Red arrow entering a copy symbol, followed by a black arrow.
  - **BPROP**: Black arrow leaving a copy symbol, followed by a red arrow.
Plan for Today

• Automatic Differentiation
  – Patterns in backprop
  – Jacobians in FC+ReLU NNs
Backprop
Jacobian of ReLU

\[ g(x) = \max(0, x) \]

(layer \( l \))

4096-d input vector

4096-d output vector

\[ \nabla^{l-1}_h \in \mathbb{R}^{4096} \]

\[ \nabla^l_h = \max(0, \nabla^{l-1}_h) \]

\[ \left[ \frac{\partial h^l}{\partial h^{l-1}} \right]_{4096 \times 4096} \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Jacobian of ReLU

Q: what is the size of the Jacobian matrix?

\[ g(x) = \max(0, x) \] (elementwise)
Jacobian of ReLU

$g(x) = \max(0, x)$ (elementwise)

Q: what is the size of the Jacobian matrix? [4096 x 4096!]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Q: what is the size of the Jacobian matrix? [4096 x 4096!]

\[ g(x) = \max(0, x) \] (elementwise)

4096-d input vector

4096-d output vector

J = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}

Q2: what does it look like?

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Jacobians of FC-Layer
Jacobians of FC-Layer

\[ \frac{\partial L}{\partial \hat{x}^{(e-1)}} \]

\[ \hat{x}^{(e-1)} = W \hat{h} \]

\[ \frac{\partial L}{\partial \hat{x}^{(e-1)}} = \begin{bmatrix} \frac{\partial L}{\partial \hat{h}} & \frac{\partial L}{\partial \hat{h}} \end{bmatrix} \left[ \frac{\partial \hat{h}}{\partial \hat{x}^{(e-1)}} \right] \]

\[ \text{input} \cdot W \]
Jacobians of FC-Layer
Jacobians of FC-Layer