CS 4803 / 7643: Deep Learning

Topics:
- (Finish) Computing Gradients
  - Backprop in Conv Layers
- Forward mode vs Reverse mode AD
- Modern CNN Architectures

Zsolt Kira
Georgia Tech
Administrivia

• HW1 due date moved!
  – Due: 02/18, 11:55pm

• Project topic submissions
  – Submit by 02/21 to get comments
  – Form filled out with:
    • Members identified
    • Paragraph of problem and another paragraph of what has been done in the literature and approach (note: the approach can be selected from an existing paper and reimplemented)
    • Description of what each member will do
    • Link
  – A project idea: ICLR reproducibility challenge
    (https://reproducibility-challenge.github.io/iclr_2019/)
    • Official submission Jan but can still do it and submit later!
• Google cloud credits out! (see piazza for details)

• Clouderizer ties in with Google Cloud Platform (GCP)
Matrix/Vector Derivatives Notation

\[
\begin{align*}
S & \quad V & \quad M \\
\frac{\partial y}{\partial x} & \quad \frac{\partial y}{\partial x} & \quad \frac{\partial y}{\partial x} \\
\frac{\partial y}{\partial x} & \quad \frac{\partial y}{\partial x} & \quad \frac{\partial y}{\partial x} \\
M & \quad \frac{\partial y}{\partial x} & \quad \text{Tensor}
\end{align*}
\]

Convention:

\[
\frac{\partial y}{\partial x} = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} \\
\vdots \\
\frac{\partial y_k}{\partial x_k}
\end{bmatrix}
\]

- Numerator = \(\text{dim} \ 1\)
- Denominator = \(\text{col-vector}\)

\[
\text{Gradient} \quad \frac{\partial y}{\partial x} = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} \\
\vdots \\
\frac{\partial y_k}{\partial x_k}
\end{bmatrix}
\]

- Denominator = \(\text{dim} \ 2\)
- Row-vector
Example: \( \ell_i (\mathbf{w}) = -\log \left( \frac{1}{1 + e^{-x^T \mathbf{w}}} \right) \) \[ \text{For } y_i = +1 \]

\[
\ell_i (\mathbf{w}) = -\log \left( \frac{\hat{y}_i}{1 - \hat{y}_i} \right)
\]

\[
\frac{\partial \ell_i}{\partial \mathbf{w}} = - \left[ \frac{1}{\hat{y}_i (1 - \hat{y}_i)} \right] . \left[ x_i \right] . \left[ -e^{x^T \mathbf{w}} \right] = (1 - \hat{y}_i) x^T
\]
Path 1: $\frac{\partial E}{\partial x_{i-1}}$ from $\frac{\partial E}{\partial x_i}$

Path 2: $\frac{\partial E}{\partial w_i}$ from $\frac{\partial E}{\partial x_i}$
Plan for Today

- Topics:
  - (Finish) Computing Gradients
    - Backprop in Conv Layers
  - Forward mode vs Reverse mode AD
  - Modern CNN Architectures
Backprop in Convolutional Layers
How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    • aka “backprop”
Computational Graph

\[ f = W x \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Any DAG of differentiable modules is allowed!
Directed Acyclic Graphs (DAGs)

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

- Concept
  - Topological Ordering
Directed Acyclic Graphs (DAGs)
Numerical vs Analytic Gradients

\[ \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

**Numerical gradient**: slow :(, approximate :(, easy to write :)

**Analytic gradient**: fast :), exact :), error-prone :(  

In practice: Derive analytic gradient, check your implementation with numerical gradient. This is called a **gradient check**.
How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”
Forward mode vs Reverse Mode

• Key Computations
Forward mode AD

g
Reverse mode AD
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

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

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

\[ \dot{w}_1 = \cos(x_1) \dot{x}_1 \quad \dot{w}_2 = \dot{x}_1 x_2 + x_1 \dot{x}_2 \]
Example: Forward mode AD

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

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

\[ \dot{w}_1 = \cos(x_1) \dot{x}_1 \quad \dot{w}_2 = \dot{x}_1 x_2 + x_1 \dot{x}_2 \]
Example: Reverse mode AD

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

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

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

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

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

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

(C) Dhruv Batra and Zsolt Kira 26
Forward mode vs Reverse Mode

- What are the differences?

\[
\begin{align*}
\dot{w}_3 &= \dot{w}_1 + \dot{w}_2 \\
\dot{w}_1 &= \cos(x_1)\dot{x}_1 \\
\dot{w}_2 &= \dot{x}_1 x_2 + x_1 \dot{x}_2 \\
\sin() &
\end{align*}
\]

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

• What are the differences?

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

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

**add** gate: gradient distributor
add gate: gradient distributor

Q: What is a max gate?

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

**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

FPROP

SUM

COPY

BPROP

COPY

(C) Dhruv Batra and Zsolt Kira
Modularized implementation: forward / backward API

Graph (or Net) object  *(rough psuedo 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 are scalars)
Example: Caffe layers

Caffe is licensed under BSD 2-Clause
Caffe Sigmoid Layer

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

\[
(1 - \sigma(x)) \sigma(x)
\]

* top_diff (chain rule)
The diagram shows a sequence of functions and inputs. The variables are as follows:

- $x_0$ is the initial input.
- $f_1$, $f_2$, ..., $f_L$ are the functions applied sequentially.
- The variables $x_1$, $x_2$, ..., $x_{L-1}$ are the outputs of each function.
- $w_1$, $w_2$, ..., $w_L$ are the weights associated with each function.
- The final output is $x_L \in \mathbb{R}$.