CS 4803 / 7643: Deep Learning

Topics:
- (Finish) Computing Gradients
  - Backprop in FC+ReLU NNs
- Convolutional Neural Networks

Dhruv Batra
Georgia Tech
Administrativia

• HW1 Reminder
  – Due: 10/02, 11:55pm

• HW0 grades
  – Out week of 10/01
Project

• Goal
  – Chance to try Deep Learning
  – **Combine with other classes / research / credits / anything**
    • You have our blanket permission
    • Extra credit for shooting for a publication
    • Get permission from other instructors; delineate different parts
  – Encouraged to apply to your research (computer vision, NLP, robotics,…)
  – Must be done this semester.
  – Groups of 3-4

• Main categories
  – Application/Survey
    • Compare a bunch 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
Computing

• Major bottleneck
  – GPUs

• Options
  – Your own / group / advisor’s resources
  – Google Cloud Credits
    • $50 credits to every registered student courtesy Google
  – https://colab.research.google.com
  – Minsky cluster in IC
Administrativia

- Project Teams Google Doc
  - https://docs.google.com/spreadsheets/d/1BipWLvvWb7Fu6OSDd-uOCF1Lr_4drKOCRVdhxm_eSHc/edit#gid=0
  - 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”
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

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

- **Concept**
  - Topological Ordering

\[ \exists \sigma: V \rightarrow \{1, \ldots, n\} \]
\[ s.t. \ (v_i, v_j) \in E \Rightarrow \sigma(v_i) < \sigma(v_j) \]
Directed Acyclic Graphs (DAGs)
Computational Graphs

• Notation

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

\[ \frac{\partial \mathbf{h}^e}{\partial \mathbf{h}^1} = \begin{bmatrix} J_{ge} & 0 & \cdots & 0 \\ 0 & J_{ge-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & J_{g2} \end{bmatrix} \]

\[ = \sum_{x} \int_{x} \int_{x} \left[ \frac{\partial \mathbf{h}^e}{\partial \mathbf{h}^1} \right]_{x} \, dx \, dx \]

\[ \approx O(\alpha d^2) \]
Forward mode AD

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

\[ \frac{\partial L}{\partial \mathbf{h}} = \frac{\partial L}{\partial \mathbf{g}} \cdot \frac{\partial \mathbf{g}}{\partial \mathbf{h}} \]

input

\[ \mathbf{g}(\cdot) \]

\[ \mathbf{h} \]
Forward mode vs Reverse Mode

- Key Computations

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial x_i} \cdot \frac{\partial x_i}{\partial x}
\]
Example: Forward mode AD

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

\[ \frac{df}{dx} = \begin{bmatrix} \frac{df}{dx_1} & \frac{df}{dx_2} \end{bmatrix} = \begin{bmatrix} x_2 & x_1 \\ \cos(x_1) & 0 \end{bmatrix} \]

\[ \dot{x}_4 = \frac{dx_1}{da} \]
Example: Forward mode AD

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

\[ \frac{df}{dx} \]

\[ \begin{align*}
\dot{w}_1 &= \cos(x_1) \dot{x}_1 \\
\dot{w}_2 &= \dot{x}_1 x_2 + x_1 \ddot{x}_2 \\
\dot{w}_3 &= \dot{w}_1 + \dot{w}_2
\end{align*} \]
Example: Forward mode AD

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

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

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

\[ \dot{x}_1 = \dot{x}_1 \quad \dot{x}_2 = \dot{x}_2 \quad \dot{w}_3 = 1 \]

\[ \dot{w}_1 = \cos(x_1) \dot{x}_1 \quad \dot{w}_2 = x_1 \dot{x}_2 + x_1 \dot{x}_2 \quad \dot{w}_3 = \dot{w}_1 + \dot{w}_2 \]

\[ \bar{x}_1 = w_1 \cos(x_1) \quad \bar{x}_1 = w_2 x_2 \quad \bar{x}_2 = w_2 x_1 \]

\[ \bar{w}_1 = w_3 \quad \bar{w}_2 = w_3 \]

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

\[ f(. . .) = \max \left( x y + \max \{z, w \} \right) \]
Patterns in backward flow

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

\[ w_3 = w_1 + w_2 \]

\[ \frac{\partial w_3}{\partial w_1} = 1 \]

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

**add gate:** gradient distributor

Q: What is a **max** gate?
Patterns in backward flow

**add** gate: gradient distributor

**max** gate: gradient router

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Q: What is a mul gate?

Patterns in backward flow

**add** gate: gradient distributor

**max** gate: gradient router
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
Duality in Fprop and Bprop
Plan for Today

- (Finish) Computing Gradients
  - Backprop in FC+ReLU NNs

- Convolutional Neural Networks
Jacobian of ReLU

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

]-dimensional input vector

4096-d output vector

\( h^{\text{e-1}} \in \mathbb{R}^{4096} \)

\( h^{0} \in \mathbb{R}^{4096} \)
Jacobian of ReLU

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

(Elementwise)

Q: what is the size of the Jacobian matrix?

\[
\frac{\partial h^i}{\partial h^j} = \begin{cases} 
1 & \text{if } i = j \\
0 & \text{otherwise}
\end{cases}
\]

\[ 4096 \times 4096 \]

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

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

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

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

In practice we process an entire minibatch (e.g. 100) of examples at one time:

i.e. Jacobian would technically be a [409600 x 409600] matrix :/

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

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

Q2: what does it look like?

\[ g(x) = \max(0, x) \quad (\text{elementwise}) \]

4096-d input vector

4096-d output vector

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

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

(elementwise)

4096-d input vector

4096-d output vector

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

\[ \frac{\partial L}{\partial \mathbf{W}^l} = \frac{\partial L}{\partial \mathbf{h}^l} \cdot \frac{\partial \mathbf{h}^l}{\partial \mathbf{W}^l} \cdot \mathbf{F}_{\text{Pass}} \]

\[ \mathbf{h}^l = \mathbf{W}^l \mathbf{h}^{l-1} \]
Jacobians of FC-Layer
Jacobians of FC-Layer
Jacobians of FC-Layer
Convolutional Neural Networks
(without the brain stuff)
- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Example: 200x200 image
40K hidden units

~2B parameters!!!

Slide Credit: Marc'Aurelio Ranzato
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

STAIONARITY? Statistics is similar at different locations

Slide Credit: Marc'Aurelio Ranzato
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Convolutional Layer

Slide Credit: Marc'Aurelio Ranzato
Convolutions!

math $\rightarrow$ CS $\rightarrow$ programming
Convolutions for mathematicians

\[ y(t) = (x \ast w)(t) = (w \ast x)(t) \]

\[ y(t) = \int_{-\infty}^{\infty} x(t - \alpha)w(\alpha) \, d\alpha \]

\[ y(t) = \int_{-\infty}^{\infty} x(\alpha)w(t - \alpha) \, d\alpha \]
Convolutions for mathematicians

\[ w(a) \rightarrow w(-a) \]

\[ w(-a) \rightarrow w(-(a+t)) \]

\[ \int_{-\infty}^{\infty} x(t) w(t-a) \]

\[ \Rightarrow y(t) = \int_{-\infty}^{\infty} x(t) w(t-a) \]
\[ y(t_1, t_2) = \int_{-\infty}^{\infty} x(t, -a) \cdot t_2 - b \, \mathrm{d}a \cdot \mathrm{d}b \]
Convolutions for computer scientists

\[ y[r, c] = \sum_{a=-\infty}^{\infty} \sum_{b=-\infty}^{\infty} x[r+a, c-b]w[a, b] \]
Convolutions for computer scientists
Convolutions for programmers

\[ y[x, c] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} x[(x+a, c+b)] w[a, b] \]
Convolutions for programmers
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer

Mathieu et al. “Fast training of CNNs through FFTs” ICLR 2014
Convolution Explained

- [https://github.com/bruckner/deepViz](https://github.com/bruckner/deepViz)
Convolutional Layer

Convolutional layer operation:

$$\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix} \ast \begin{bmatrix}
\_ & \_ & \_ \\
\_ & \_ & \_ \\
\_ & \_ & \_ \\
\end{bmatrix} = \begin{bmatrix}
\_ & \_ & \_ \\
\_ & \_ & \_ \\
\_ & \_ & \_ \\
\end{bmatrix}$$
Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
Convolutional Layer
Convolutional Layer
Convolution Layer

32x32x3 image -> preserve spatial structure

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

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

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

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Filters always extend the full depth of the input volume

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

32x32x3 image
5x5x3 filter \( w \)

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. \( 5 \times 5 \times 3 = 75 \)-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map

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

32x32x3 image
5x5x3 filter
convolve (slide) over all spatial locations

consider a second, green filter

activation maps

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!