CS 4803 / 7643: Deep Learning

Topics:
- (Finish) Automatic Differentiation
- Patterns in backprop
- Jacobians in FC+ReLU NNs

Dhruv Batra
Georgia Tech
Administrativia

• HW2 out
  – Due: 09/23 11:59pm
  
  – Theory: Gradient descent, Hessians, Auto-diff, Convolutions
    • https://www.overleaf.com/project/5f4c143e06061d00013dd4f0
  
  – Implementation: ConvNets in Python and PyTorch
  
  – Bonus: Challenge on EvalAI
    • https://evalai.cloudcv.org/web/challenges/challenge-page/684/overview
Project

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

• Main categories
  – Reproducibility
    • Pick a paper from a recent conference. Attempt to reproduce the method and validate claims.
  – 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.
ML Reproducibility Challenge 2020

Welcome to the ML Reproducibility Challenge 2020! This is already the fourth edition of this event (see V1, V2, V3), and we are excited this year to announce that we are broadening our coverage of conferences and papers to cover several new top venues, including: NeurIPS, ICML, ICLR, ACL, EMNLP, CVPR and ECCV.

The primary goal of this event is to encourage the publishing and sharing of scientific results that are reliable and reproducible. In support of this, the objective of this challenge is to investigate reproducibility of papers accepted for publication at top conferences by inviting members of the community at large to select a paper, and verify the empirical results and claims in the paper by reproducing the computational experiments, either via a new implementation or using code/data or other information provided by the authors.

All submitted reports will be peer reviewed and shown next to the original papers on Papers with Code. Reports will be peer-reviewed via OpenReview. Every year, a small number of these reports, selected for their clarity, thoroughness, correctness and insights, are selected for publication in a special edition of the journal ReScience. (see J1, J2).
TAs

Sameer Dharur
Joanne Truong
Yihao Chen
Michael Piseno
Hrishikesh Kale
Tianyu Zhan
Prabhav Chawla
Guillermo Nicolas Grande
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-notebook + free GPU instance
Administrativia

• Project Teams
  – [https://gtvault-my.sharepoint.com/:x:/g/personal/dbatra8_gatech_edu/EY4_65XOzWtOkXSSz2WgpoUBY8ux2gY9PsRzR6KnglIFEQ?e=4tnKWI](https://gtvault-my.sharepoint.com/:x:/g/personal/dbatra8_gatech_edu/EY4_65XOzWtOkXSSz2WgpoUBY8ux2gY9PsRzR6KnglIFEQ?e=4tnKWI)
  – Project Title
  – 1-3 sentence project summary TL;DR
  – Team member names

(C) Dhruv Batra
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

Goal: $\frac{\partial L}{\partial x_1}$

Layer $l$

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

Input: $\frac{\partial h^{(l-1)}}{\partial x}$

$\frac{\partial h^e}{\partial x} = \begin{bmatrix} \frac{\partial h^e}{\partial h^{(l-1)}} & \frac{\partial h^e}{\partial h^{(l-2)}} \end{bmatrix}$

Layer Jacobin

Input

FM-AD
Reverse mode AD

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

Output: \[ \mathbf{h} = g(\mathbf{h}^{-1}) \]

Input: \[ \frac{\partial L}{\partial \mathbf{h}} \]

Layer Jacobian
Example: Forward mode AD

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

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

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

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

\[ f(x_1, x_2) = \sin(x_1) + x_1 x_2 \]
Forward mode vs Reverse Mode

- $x \xrightarrow{\text{Graph}} L$
- Intuition of Jacobian

\[
\frac{\partial L}{\partial x} = \begin{bmatrix}
\frac{\partial L}{\partial x_1} \\
\frac{\partial L}{\partial x_2} \\
\vdots \\
\frac{\partial L}{\partial x_n}
\end{bmatrix} = \begin{bmatrix}
j & \frac{\partial L}{\partial x_1} & \frac{\partial L}{\partial x_2} & \cdots & \frac{\partial L}{\partial x_n} \\
& & & & \\
& & & & \\
& & & & \\
& & & & \\
& & & & \\
& & & & \\
& & & & \\
& & & & \\
\end{bmatrix}
\]

for $i$ in $1: C$

- Reverse Mode AD output in 1 pass

- Forward Mode AD output in 1 pass
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?
Neural Network Computation Graph

$\hat{x}_i = \text{max}(0, h_i x)$

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

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

(C) Dhruv Batra

Figure Credit: Andrea Vedaldi
Backprop
Key Computation: Forward-Prop

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

\[
\frac{\partial L}{\partial X} \left\{ \frac{\partial Z}{\partial X}, \frac{\partial Z}{\partial \theta} \right\} \frac{\partial L}{\partial Z}
\]
Neural Network Training

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

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

• Step 1: Compute Loss on mini-batch [F-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 \textit{wrt} parameters [B-Pass]

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]
- Step 3: Use gradient to update parameters

\[
\theta \leftarrow \theta - \eta \frac{dL}{d\theta}
\]
Any DAG of differentiable modules is allowed!
Plan for Today

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

(C) Dhruv Batra
Backpropagation: a simple example

\[ f(x, y, z, w) = 2 \left( \frac{x y}{w_1} + \frac{\max\{z, w_2\}}{w_2} \right) \]
Forward pass
Patterns in backprop

\[ f = 2w_3 \]

\[ w_3 = \frac{df}{dw_3} \]

\[ f = \frac{df}{df} = 1 \]

\[ = 2 \]
Q: What is an add gate?

\[ w_2 = w_1 + w_2 \]
Patterns in backprop

**add gate:** gradient distributor
Q: What is a max gate?

\[ z = \frac{\partial f}{\partial z} = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix} \]

\[
\frac{\partial w_1}{\partial z} = \begin{cases} +1 & \text{if } z > w \\ 0 & \text{if } z < w \\ \text{not defined} & \text{if } z = w \end{cases}
\]

\[ w_2 = \max\{z, w\} = \begin{cases} z & \text{if } z > w \\ w \end{cases} \]
Patterns in backprop

\textbf{add} gate: gradient distributor

\textbf{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 backprop

- **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
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)

```python
class MultiplyGate(object):
    def forward(x, y):
        z = x*y
        return z
    def backward(dz):
        dx = ... #todo
        dy = ... #todo
        return [dx, dy]
```

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Modularized implementation: forward / backward API

(x,y,z are scalars)

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

Caffe is licensed under BSD 2-Clause.

---

Caffe is licensed under BSD 2-Clause.
Caffe Sigmoid Layer

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

* top_diff (chain rule)

\[ (1 - \sigma(x)) \sigma(x) \]
Plan for Today

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

\[ \mathbf{x} \xrightarrow{f_1} \mathbf{h}^{(1)} \xrightarrow{f_2} \mathbf{h}^{(2)} \cdots \xrightarrow{f_{L-1}} \mathbf{h}^{(L-1)} \xrightarrow{f_L} \mathbf{L} \]

\[ \mathbf{dx} \xleftarrow{d\mathbf{f}_1} \mathbf{dh}^{(1)} \xleftarrow{d\mathbf{f}_2} \mathbf{dh}^{(2)} \cdots \xleftarrow{d\mathbf{f}_{L-1}} \mathbf{dh}^{(L-1)} \xleftarrow{d\mathbf{f}_L} \mathbf{dL} \]

\[ \mathbf{dw}_1 \xrightarrow{d\mathbf{f}_1} \mathbf{dw}_2 \xrightarrow{d\mathbf{f}_2} \mathbf{dw}_{L-1} \xrightarrow{d\mathbf{f}_{L-1}} \mathbf{dw}_L \]
Jacobian of ReLU

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

4096-d input vector \rightarrow \text{ReLU} \rightarrow 4096-d output vector

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

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

4096-d input vector \rightarrow g(x) \rightarrow 4096-d output vector

Q1: what is the size of the Jacobian matrix?
g(x) = max(0, x) [elementwise]

Q1: what is the size of the Jacobian matrix? [4096 x 4096]
i.e. Jacobian would technically be a matrix:\n
\[
g(x) = \max(0, x) \quad \text{(elementwise)}
\]

4096-d input vector \quad g(x) = \max(0, x) \quad \text{(elementwise)} \quad \text{4096-d output vector}

Q1: 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 [409,600 x 409,600] matrix :\n
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Jacobian of ReLU

Q1: what is the size of the Jacobian matrix? [4096 x 4096!]

Q2: what does it look like?

\[
g(x) = \max(0, x) \quad \text{(elementwise)}
\]
Jacobians of FC-Layer
Jacobians of FC-Layer
Jacobians of FC-Layer
Jacobians of FC-Layer