Topics:
  – Specifying Layers
  – Forward & Backward autodifferentiation
  – (Beginning of) Convolutional neural networks

Zsolt Kira
Georgia Tech
Administrivia

• PS0 released
  – mean of 20.7
  – standard deviation of 3.4
  – median of 21
  – max of 25
  – See me if you did not pass

• PS1/HW1 out

• Start thinking about project topics/teams
  – More details on project next time
Recap from last time
Some design decisions:

- How many examples to use to calculate gradient per iteration?
- What should alpha (learning rate) be?
  - Should it be constant throughout?
- How many epochs to run to?
Any DAG of differentiable modules is allowed!
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 [F-Pass]
- Step 2: Compute gradients wrt parameters [B-Pass]
General Flow Graphs

Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

\[ \{y_1, y_2, \ldots, y_n\} = \text{successors of } x \]

\[
\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \left( \frac{\partial z}{\partial y_i} \cdot \frac{\partial y_i}{\partial x} \right)
\]

ReLU

Forward pass

\[ h_{l+1} = \max \{ 0, h_{l} \} \]

Backwards pass

\[ \frac{\partial L}{\partial h_{l}} = \begin{cases} 1 & \text{if } h_{l} \geq 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ \frac{\partial L}{\partial h_{l+1}} = \frac{\partial L}{\partial h_{l}} \cdot \frac{\partial h_{l+1}}{\partial h_{l}} \]

\[ \frac{\partial L}{\partial h_{l+1}} = \frac{2L}{2h_{l}} \cdot \frac{2h_{l+1}}{2h_{l}} = \frac{2L}{2h_{l+1}} \cdot \left\{ h_{l+1} \geq 0 \right\} \]
Jacobian of ReLU

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

4096-d input vector \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad
Jacobian of ReLU

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

Q: what is the size of the Jacobian 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!]

\[ g(x) = \max(0, x) \] (elementwise)
Q: what is the size of the Jacobian matrix? [4096 x 4096!]

Q2: what does it look like?

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

4096-d input vector

4096-d output vector

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Plan for Today

• Specifying Layers

• Forward & Backward auto-differentiation

• (Beginning of) Convolutional neural networks
Deep Learning = Differentiable Programming

- **Computation = Graph**
  - Input = Data + Parameters
  - Output = Loss
  - Scheduling = Topological ordering

- **What do we need to do?**
  - Generic code for representing the graph of modules
  - Specify modules (both forward and backward function)
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
```

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

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

```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]
```

(x, y, z are scalars)
Example: Caffe layers

Caffe is licensed under BSD 2-Clause
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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 vs Reverse Mode

- Key Computations
Forward mode AD
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

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

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

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

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

Q: What happens if there's another input variable \( x_3 \)?
A: more sophisticated graph; d “forward props” for d variables

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[ \bar{x}_1 = \bar{w}_1 \cos(x_1) \]
\[ \bar{x}_1 = \bar{w}_2 x_2 \]
\[ \bar{x}_2 = \bar{w}_2 x_1 \]
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 \( f_2 \)?

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

\[ f(x_1, x_2) = x_1x_2 + \sin(x_1) \]

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

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

\[
\begin{align*}
\ddot{x}_1 &= \sin(x_1) \\
\ddot{x}_2 &= \dot{x}_1 \ddot{x}_2 + x_1 \sin(x_1) \\
\ddot{w}_1 &= \ddot{w}_3 \\
\ddot{w}_2 &= \ddot{w}_3 \ddot{x}_2 + \ddot{x}_1 \sin(x_1) \\
\ddot{w}_3 &= 1 \\
\end{align*}
\]
Practical Note 2: Software Frameworks

Caffe
(UC Berkeley)

Caffe2
(Facebook)

A few weeks ago!

Torch
(NYU / Facebook)

PyTorch
(Facebook)

Paddle
(Baidu)

CNTK
(Microsoft)

Theano
(U Montreal)

TensorFlow
(Google)

+Keras

MXNet
(Amazon)

And others...
## PyTorch

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>torch</td>
<td>a Tensor library like NumPy, with strong GPU support</td>
</tr>
<tr>
<td>torch.autograd</td>
<td>a tape based automatic differentiation library that supports all differentiable Tensor operations in torch</td>
</tr>
<tr>
<td>torch.nn</td>
<td>a neural networks library deeply integrated with autograd designed for maximum flexibility</td>
</tr>
<tr>
<td>torch.optim</td>
<td>an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.</td>
</tr>
<tr>
<td>torch.multiprocessing</td>
<td>python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and hogwild training.</td>
</tr>
<tr>
<td>torch.utils</td>
<td>DataLoader, Trainer and other utility functions for convenience</td>
</tr>
<tr>
<td>torch.legacy(nn/.optim)</td>
<td>legacy code that has been ported over from torch for backward compatibility reasons</td>
</tr>
</tbody>
</table>
A graph is created on the fly

```python
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```
Plan for Today (Cont.)

• Specifying Layers

• Forward & Backward auto-differentiation

• (Beginning of) Convolutional neural networks
  – What is a convolution?
  – FC vs Conv Layers
Recall: Linear Classifier

\[ f(x, W) = Wx + b \]

Array of 32x32x3 numbers (3072 numbers total)

Parameters or weights

10 numbers giving class scores

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

Stretch pixels into column

\[ W \]

\[ b \]

\[
\begin{array}{cccc}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3 \\
\end{array}
\]

\[
\begin{array}{c}
56 \\
231 \\
24 \\
2 \\
\end{array}
\]

\[
\begin{array}{c}
56 \\
231 \\
24 \\
2 \\
\end{array}
\]

\[
\begin{array}{c}
1.1 \\
3.2 \\
-1.2 \\
-96.8 \\
437.9 \\
61.95 \\
\end{array}
\]

Cat score
Dog score
Ship score

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Recall: (Fully-Connected) Neural networks

(Before) Linear score function:
\[ f = Wx \]

(Now) 2-layer Neural Network
\[ f = W_2 \max(0, W_1x) \]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Convolutional Neural Networks

(without the brain stuff)
Fully Connected Layer

Example: 200x200 image
40K hidden units

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

~2B parameters!!!
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

STATIONARITY?
Statistics similar at all locations
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels
What filter to use?

In mathematics and, in particular, functional analysis, **convolution** is a mathematical operation on two functions $f$ and $g$, producing a third function that is typically viewed as a modified version of one of the original functions, giving the area overlap between the two functions as a function of the amount that one of the original functions is translated. Convolution is similar to **cross-correlation**. It has applications that include probability, statistics, computer vision, image and signal processing, electrical engineering, and differential equations.
Discrete convolution

• Discrete Convolution!
  • Very similar to correlation but associative

\[ F \otimes (G \otimes I) = (F \otimes G) \otimes I \]

1D Convolution

\[ y_k = \sum_{n=0}^{N-1} h_n \cdot x_{k-n} \]

\[ \begin{align*}
y_0 &= h_0 \cdot x_0 \\
y_1 &= h_1 \cdot x_0 + h_0 \cdot x_1 \\
y_2 &= h_2 \cdot x_0 + h_1 \cdot x_1 + h_0 \cdot x_2 \\
y_3 &= h_2 \cdot x_1 + h_1 \cdot x_2 + h_0 \cdot x_3 \\
&\vdots
\end{align*} \]

2D Convolution

\[ H[m,n] = f \otimes I = \sum_{k,l} f[k,l] I[m-k,n-l] \]

Filter
A note on sizes

MATLAB to the rescue!
• \texttt{conv2(x, w, 'valid')}
Convolutions!

- Math vs. CS vs. programming viewpoints
Convolutions for mathematicians

• On operation on two functions $x$ and $w$ to produce a third function $y$
• E.g. input $x(t)$ and kernel or weighting function $w(\tau)$

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

• One dimension

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

• Two dimensions

\[ y(t_1, t_2) = (x * w)(t_1, t_2) = \int_{\tau=-\infty}^{\infty} \int_{\sigma=-\infty}^{\infty} x(t_1 - \tau, t_2 - \sigma)w(\tau, \sigma) d\tau d\sigma \]
Convolutions for CS/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
Convolution Explained

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

Learn multiple filters.

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

32x32x3 image -> preserve spatial structure
Convolution Layer

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

Filters always extend the full depth of the input volume

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

consider a second, **green** filter

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

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!
Im2Col

Figure Credit: https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
General Matrix Multiply (GEMM)

Figure Credit: https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
Time Distribution of AlexNet
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

![Diagram](slide-content)

- CONV, ReLU
- e.g. 6 5x5x3 filters

*Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n*
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.