Backpropagation
Reminders

• Homework 4: Logistic Regression
  – Out: Fri, Feb 15
  – Due: Fri, Mar 1 at 11:59pm

• Homework 5: Neural Networks
  – Out: Fri, Mar 1
  – Due: Fri, Mar 22 at 11:59pm

• Today’s In-Class Poll
  – http://p13.mlcourse.org
  – Also linked from Schedule page on mlcourse.org
<table>
<thead>
<tr>
<th>Q:</th>
<th>What is mini-batch SGD?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A:</td>
<td>A variant of SGD...</td>
</tr>
</tbody>
</table>
Mini-Batch SGD

• **Gradient Descent:**
  Compute true gradient exactly from all $N$ examples

• **Mini-Batch SGD:**
  Approximate true gradient by the average gradient of $K$ randomly chosen examples

• **Stochastic Gradient Descent (SGD):**
  Approximate true gradient by the gradient of one randomly chosen example
Mini-Batch SGD

while not converged: \( \theta \leftarrow \theta - \lambda g \)

Three variants of first-order optimization:

Gradient Descent: \( g = \nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla J^{(i)}(\theta) \)

SGD: \( g = \nabla J^{(i)}(\theta) \) \text{ where } i \text{ sampled uniformly}

Mini-batch SGD: \( g = \frac{1}{S} \sum_{s=1}^{S} \nabla J^{(i_s)}(\theta) \) \text{ where } i_s \text{ sampled uniformly } \forall s \)
Computing Gradients

DIFFERENTIATION
1. Given training data:
   \[ \{x_i, y_i\}_{i=1}^N \]

2. Choose each of these:
   - Decision function
     \[ \hat{y} = f_\theta(x_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:
   \[ \theta^* = \arg\min_{\theta} \sum_{i=1}^N \ell(f_\theta(x_i), y_i) \]

4. Train with SGD:
   (take small steps opposite the gradient)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i) \]
Approaches to Differentiation

• **Question 1:**
  When can we compute the gradients for an arbitrary neural network?

• **Question 2:**
  When can we make the gradient computation efficient?
Approaches to Differentiation

1. Finite Difference Method
   - Pro: Great for testing implementations of backpropagation
   - Con: Slow for high dimensional inputs / outputs
   - Required: Ability to call the function $f(x)$ on any input $x$

2. Symbolic Differentiation
   - Note: The method you learned in high-school
   - Note: Used by Mathematica / Wolfram Alpha / Maple
   - Pro: Yields easily interpretable derivatives
   - Con: Leads to exponential computation time if not carefully implemented
   - Required: Mathematical expression that defines $f(x)$

3. Automatic Differentiation - Reverse Mode
   - Note: Called Backpropagation when applied to Neural Nets
   - Pro: Computes partial derivatives of one output $f(x)_i$ with respect to all inputs $x_j$ in time proportional to computation of $f(x)$
   - Con: Slow for high dimensional outputs (e.g. vector-valued functions)
   - Required: Algorithm for computing $f(x)$

4. Automatic Differentiation - Forward Mode
   - Note: Easy to implement. Uses dual numbers.
   - Pro: Computes partial derivatives of all outputs $f(x)_i$ with respect to one input $x_j$ in time proportional to computation of $f(x)$
   - Con: Slow for high dimensional inputs (e.g. vector-valued $x$)
   - Required: Algorithm for computing $f(x)$

Given $f : \mathbb{R}^A \rightarrow \mathbb{R}^B, f(x)$

Compute $\frac{\partial f(x)_i}{\partial x_j} \forall i, j$
The centered finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{(J(\theta + \epsilon \cdot d_i) - J(\theta - \epsilon \cdot d_i))}{2\epsilon}$$

(1)

where $d_i$ is a 1-hot vector consisting of all zeros except for the $i$th entry of $d_i$, which has value 1.

Notes:

• Suffers from issues of floating point precision, in practice

• Typically only appropriate to use on small examples with an appropriately chosen epsilon
Differentiation Quiz #1:
Suppose \(x = 2\) and \(z = 3\), what are \(dy/dx\) and \(dy/dz\) for the function below? **Round your answer to the nearest integer.**

\[
y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{xz}
\]

**Answer:** Answers below are in the form \([dy/dx, dy/dz]\)

A. [42, -72]  
B. [72, -42]  
C. [100, 127]  
D. [127, 100]  
E. [1208, 810]  
F. [810, 1208]  
G. [1505, 94]  
H. [94, 1505]
Differentiation Quiz #2:

A neural network with 2 hidden layers can be written as:

$$y = \sigma(\beta^T \sigma((\alpha^{(2)})^T \sigma((\alpha^{(1)})^T x)))$$

where $y \in \mathbb{R}$, $x \in \mathbb{R}^{D^{(0)}}$, $\beta \in \mathbb{R}^{D^{(2)}}$ and $\alpha^{(i)}$ is a $D^{(i)} \times D^{(i-1)}$ matrix. Nonlinear functions are applied elementwise:

$$\sigma(a) = [\sigma(a_1), \ldots, \sigma(a_K)]^T$$

Let $\sigma$ be sigmoid: $\sigma(a) = \frac{1}{1+exp(-a)}$

What is $\frac{\partial y}{\partial \beta_j}$ and $\frac{\partial y}{\partial \alpha^{(i)}_j}$ for all $i, j$. 

[Diagram of a neural network with inputs $x_1, x_2, x_3, \ldots, x_M$, hidden layers, and output $y$.]
CHAIN RULE
Chain Rule

Chalkboard

– Chain Rule of Calculus
Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

$$
\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k
$$
Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$

Backpropagation is just repeated application of the chain rule from Calculus 101.
Intuitions

BACKPROPAGATION
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

Slide from (Stoyanov & Eisner, 2012)
Error Back-Propagation

$p(y|x^{(i)})$

Slide from (Stoyanov & Eisner, 2012)
Algorithm

BACKPROPAGATION
Differentiation Quiz #1:
Suppose \( x = 2 \) and \( z = 3 \), what are \( \frac{dy}{dx} \) and \( \frac{dy}{dz} \) for the function below? **Round your answer to the nearest integer.**

\[
y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{xz}
\]
Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation
1. Write an algorithm for evaluating the function $y = f(x)$. The algorithm defines a directed acyclic graph, where each variable is a node (i.e. the “computation graph”)
2. Visit each node in topological order.
   For variable $u_i$ with inputs $v_1,..., v_N$
   a. Compute $u_i = g_i(v_1,..., v_N)$
   b. Store the result at the node

Backward Computation
1. Initialize all partial derivatives $dy/du_j$ to 0 and $dy/dy = 1$.
2. Visit each node in reverse topological order.
   For variable $u_i = g_i(v_1,..., v_N)$
   a. We already know $dy/du_i$
   b. Increment $dy/dv_j$ by $(dy/du_i)(du_i/dv_j)$
      (Choice of algorithm ensures computing $(du_i/dv_j)$ is easy)

Return partial derivatives $dy/du_i$ for all variables