Announcements
Announcements

• Wed, usual time/place
• **not** finals period!
• Closed book, but 2 sheets of notes are allowed

• **Open-ended projects due midnight Sun 5/6**

• I fixed that quiz from last week – Tues noon
Deep Neural Networks
Generalizing backprop

- Starting point: a function of \( n \) variables
- Step 1: code your function as a series of assignments
- Step 2: back propagate by going thru the list in reverse order, starting with...

\[
\frac{dx_N}{dx_1} = \sum_{j:i \in \pi(j)} \frac{dx_N}{dx_j} \frac{\partial x_j}{\partial x_i}
\]

...and using the chain rule

**Wengert list**

- e.g.

\[
x_7 = x_2 + x_5 \\
\pi(7) = (2, 5) \\
f_7 = \text{add}
\]

**Step 1: forward**

**inputs:** \( x_1, x_2, \ldots, x_n \)

\[
\text{for } i = n + 1, n + 2, \ldots, N \\
x_i \leftarrow f_i(x_{\pi(i)})
\]

**return** \( x_N \)

**Step 2: backprop**

\[
\text{for } i = N - 1, N - 2, \ldots, 1 \\
\frac{dx_N}{dx_i} \leftarrow \sum_{j:i \in \pi(j)} \frac{dx_N}{dx_j} \frac{\partial f_j}{\partial x_i}
\]

A function

Computed in previous step

https://justindomke.wordpress.com/
Example: 2-layer neural network

Inputs: X, W1, B1, W2, B2
- Z1a = mul(X, W1) // matrix mult
- Z1b = add*(Z1a, B1) // add bias vec
- A1 = tanh(Z1b) // element-wise
- Z2a = mul(A1, W2)
- Z2b = add*(Z2a, B2)
- A2 = tanh(Z2b) // element-wise
- P = softmax(A2) // vec to vec
- C = crossEntropy(Y, P) // cost function

X is N*D, W1 is D*H, B1 is 1*H, W2 is H*K, ...

\[ p_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \]

Target Y; N examples; K outs; D feats, H hidden
Minibatch SGD on GPU

Let X be a matrix with $k$ examples
Let $w_i$ be the input weights for the $i$-th hidden unit
Then $A = X W$ is output for all $m$ units for all $k$ examples

There's a lot of chances to do this in parallel

$$XW = \begin{bmatrix} x_1 \cdot w_1 & x_1 \cdot w_2 & \cdots & x_k \cdot w_1 \\ \vdots & \vdots & \ddots & \vdots \\ x_1 \cdot w_m & x_1 \cdot w_m & \cdots & x_k \cdot w_m \end{bmatrix}$$
Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot  
DIRO, Université de Montréal, Montréal, Québec, Canada

Yoshua Bengio

Histogram of gradients in a 5-layer network for an artificial image recognition task
Gradients are unstable

Max at $1/4$

If weights are usually $< 1$ then we are multiplying by many numbers $< 1$ so the gradients get very small.

The vanishing gradient problem

What happens as the layers get further and further from the output layer? E.g., what’s gradient for the bias term with several layers after it in a trivial net?

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$
Some key differences in modern ANNs

• Use of softmax and entropic loss instead of quadratic loss.
• Use of alternate non-linearities
  — reLU and hyperbolic tangent
• Better understanding of weight initialization
• ...

[Graph showing a 3D plot with a contour plot and a surface plot.]
Bloom filters

• Interface to a Bloom filter
  – BloomFilter(int maxSize, double p);
  – void bf.add(String s); // insert s
  – bool bd.contains(String s);
    • // If s was added return true;
    • // else with probability at least 1-p return false;
    • // else with probability at most p return true;

  – I.e., a noisy “set” where you can test membership (and that’s it)
Randomized algorithms
Bloom filters

bf.add("fred flintstone"): set several "random" bits

bf.add("barney rubble"): 
Bloom filters

bf.contains ("fred flintstone"): return min of "random" bits

bf.contains ("barney rubble"):
Bloom filters

bf.contains("wilma flintstone"): 

bf.contains("wilma flintstone"): a false positive
Randomized algorithms

• What is a Bloom filter for (what’s the API)?
• What are the guarantees? What kind of errors do they make?
• How can you build up more complex operations (eg, counting to K) with multiple filters?
• How about countmin sketches?
• How about LSH?
• What are the problems that on-line LSH is trying to fix?
Architectures
Graph architectures

• Differences between
  – Signal/collect
  – GraphX
  – PowerGraph
  – GraphChi

• Can you understand/extend simple programs?

<table>
<thead>
<tr>
<th>initialState</th>
<th>if (isTrainingData) trainingData else avgProbDist</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect()</td>
<td>if (isTrainingData)</td>
</tr>
<tr>
<td></td>
<td>return oldState</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>return signals.sum.normalise</td>
</tr>
<tr>
<td>signal()</td>
<td>return source.state</td>
</tr>
</tbody>
</table>
Stale Synchronous Parallel (SSP)

LDA on NYtimes Dataset
LDA 32 machines (256 cores), 10% docs per iter

[Ho et al 2013]
LDAs and sampling
Unsupervised NB vs LDA

one class prior

one Y per doc

one Z per word

different class distrib θ for each doc

α → π → Y

β → Y

W → N_d

D

K

γ

π

β

γ

K

W

D

N_d

20
Recap: Collapsed Sampling for LDA

\[ P(z = t | w) \propto (\alpha_t + n_{t|d}) \frac{\beta + n_{w|t}}{\beta V + n_{.|t}}. \]

- \( \Pr(Z|E+) \)
- \( \Pr(E-|Z) \)

“fraction” of time \( Z=t \) in doc \( d \)

Fraction of time \( W=w \) in topic \( t \)

Only sample the Z’s

Ignores a detail – counts should not include the \( Z_{di} \) being sampled
\[ P(z = t|w) \propto (\alpha_t + n_{t|d}) \frac{\beta + n_{w|t}}{\beta V + n_{.|t}}. \]

\[ P(z = t|w) \propto \frac{\alpha_t \beta}{\beta V + n_{.|t}} + \frac{n_{t|d} \beta}{\beta V + n_{.|t}} + \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{.|t}}. \]

\[
\begin{align*}
    s &= \sum_t \frac{\alpha_t \beta}{\beta V + n_{.|t}} \\
    r &= \sum_t \frac{n_{t|d} \beta}{\beta V + n_{.|t}} \\
    q &= \sum_t \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{.|t}}.
\end{align*}
\]

\[ z = s + r + q \]
Fenwick Tree Sampler

Basic problem: how can we sample from a biased die quickly….

…and update quickly? maybe we can use a binary tree….

http://www.keithschwarz.com/darts-dice-coins/
## Data structures and algorithms

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Initialization Time</th>
<th>Initialization Space</th>
<th>Generation Time</th>
<th>Parameter Update Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSearch</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>BSearch</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(\log T)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>Alias Method</td>
<td>$O(T)$</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>F+tree Sampling</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(\log T)$</td>
<td>$O(\log T)$</td>
</tr>
</tbody>
</table>

### F+ tree

```
(\frac{23}{40}, \frac{7}{10})
```

```
(\frac{1}{4}, \frac{9}{20})
```

```
(\frac{9}{20}, \frac{23}{40})
```

```
(\frac{7}{10}, \frac{4}{5})
```

```
(\frac{9}{10}, 1)
```
Unsupervised/SS Learning on graphs

• What’s different between HF, MRW, MAD?
  – Which have hard/soft seeds?
  – How do they scale with #edges, #nodes?
• What are the methods trying to optimize?
• Do they optimize it exactly or approximately?