\[ \sum_{m=\text{Sept}}^{Dec} f(\text{cs534}) \]
MOTIVATION: MAKING EMAIL BETTER

TAGS MAKE EMAIL WORK BETTER

Processing email is time intensive and can be overwhelming.

Tags can help us retrieve email more quickly and easily, saving us time.

Machine learning makes tags simpler and easier by predicting tags for us.

GOOD ALGORITHMS HELP TAGS WORK BETTER

Feature space grows over time.

Content changes over time.

Tag space grows over time.

Tag space changes over time.

- Tags can be ephemeral, i.e. \conf\nips2014

Laptop environment is resource constrained.

Is there an algorithm that helps us deliver “good enough” performance for all of these challenges?
WHY ADA GRAD?

The promise of AdaGrad is adaptation and sparsity.

Gradient steps treat all features as equal. But they are not.

We want to utilize the infrequent features that give us predictive power.

Learning rate for each feature

Text data:

The most unsung birthday in American business and technological history this year may be the 50th anniversary of the Xerox 914 photocopier.¹

¹The Atlantic, July/August 2010.
ADAGRAD OVERVIEW

Standard stochastic (sub)gradient methods move $w_t$ in a minimizing direction, given by $-g_t$. When $X = \mathbb{R}^d$, the stochastic subgradient descent algorithm is simply:

$$w_{t+1} = w_t - \eta g_t$$

where $\eta > 0$ is a scalar learning rate.

AdaGrad provides a per-feature learning rate at each time step $t$,

$$\eta_{t,i} = \frac{\eta}{\sqrt{G_{t,ii}}}$$

Where each $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where diagonal element $i, i$ is defined to be $\sum_{t'=1}^{t} g_{t',i}^2$, that is, the sum of the squares of the $i$th dimension of all historical gradients.
ADAGRAD UPDATE WITH $\ell_1$ REGULARIZATION

Directly applying stochastic subgradient descent to an $\ell_1$ regularized objective fails to produce sparse solutions in bounded time, which has motivated several specialized algorithms that target such objectives.

We will use the AdaGrad variant of one such learning algorithm, the so-called regularized dual averaging algorithm of Xiao (2010) which makes use of the online average (sub)gradient at time \( t \).

Using the average gradient, the $\ell_1$ regularized objective may be optimized with the following update:

$$w_{t+1,i} = \begin{cases} 0, & \text{if } |\bar{g}_{t,i}| < \lambda \\ -\text{sgn}(\bar{g}_{t,i}) \frac{\eta_t}{\sqrt{G_{t,ii}}} (|\bar{g}_{t,i}| - \lambda) & \text{otherwise} \end{cases}$$

The primary role of $\eta$ is to determine how much a feature changes the very first time it is encountered, so in problems with large numbers of extremely rare features, some additional care may be warranted.

Adapted from Notes on AdaGrad (Dyer, 2013)
CLASSIFIER TESTING : DATA

10950 of my own email messages
113MB
1218 plain text
9726 HTML
6 Rich Text
50 Tags
Some common tags, many rare tags
Tag cardinality 1.13
**EXPERIMENT SETUP**

**METHOD**

Online testing – iteratively train/test/measure on single data $a$
- Represents ideal real world scenario if the end user corrected every mistake, confirmed every success, i.e. “full supervision”.

Differs from the usual train on data $a$, test/measure on data $b$ approach

Basic measurements taken at culmination of online testing, i.e. at last instance.

Mistakes = $FP + FN$

User Cost $5 = FP + (5 \times FN)$

**EXPERIMENTS**

R1 – Comparison using our default settings, i.e. dictionary filter active.

R2 – Comparison using non-filtered feature alphabet, i.e. no dictionary filter

R3 – Sparsity measurements for R1, R2

R4 – Latency, efficiency for R1, R2
ADAGRAD VS CW: BASIC METRICS

These measures don’t tell us much when we have significant class imbalance.
ROUND 1: ADAGRAD VS CW: MISTAKES, COST

CW, ADAGRAD MEASUREMENTS VS. INSTANCES

Data alphabet size 16026
ROUND 2: THIS TIME, NO DICTIONARY FILTER

Data alphabet size 51208

CW, ADAGRAD MEASUREMENTS VS. INSTANCES
## Round 3: Sparsity

### With Dictionary Filter

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Data Alphabet Size</th>
<th>Sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW</td>
<td>16026</td>
<td>1980 (12.3%)</td>
</tr>
<tr>
<td>ADAGRAD</td>
<td>16026</td>
<td>7262 (45.3%)</td>
</tr>
</tbody>
</table>

### Without Dictionary Filter

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Data Alphabet Size</th>
<th>Sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW</td>
<td>51208</td>
<td>8930 (17.4%)</td>
</tr>
<tr>
<td>ADAGRAD</td>
<td>51208</td>
<td>31428 (61.4%)</td>
</tr>
</tbody>
</table>
ROUND 4: LATENCY, EFFICIENCY

WITH DICTIONARY FILTER

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Data Alphabet Size</th>
<th>Mean Update Time(μsec)</th>
<th>Update Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW</td>
<td>16026</td>
<td>535.2</td>
<td>18742</td>
</tr>
<tr>
<td>ADAGRAD</td>
<td>16026</td>
<td>10.96</td>
<td>2368</td>
</tr>
</tbody>
</table>

WITHOUT DICTIONARY FILTER

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Data Alphabet Size</th>
<th>Mean Update Time (μsec)</th>
<th>Update Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW</td>
<td>51208</td>
<td>812.19</td>
<td>20419</td>
</tr>
<tr>
<td>ADAGRAD</td>
<td>51208</td>
<td>19.82</td>
<td>1911</td>
</tr>
</tbody>
</table>

Measurements Averaged Over 2 Complete Runs
## PERFORMANCE SUMMARY

<table>
<thead>
<tr>
<th>ROUND</th>
<th>CW</th>
<th>ADAGRAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUND 1 : ERROR RATE, USER COST (DF=T)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROUND 2 : ERROR RATE, USER COST (DF=F)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROUND 3 : SPARSITY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROUND 4 : LATENCY, EFFICIENCY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WINNER:</td>
<td></td>
<td>TIE?</td>
</tr>
</tbody>
</table>
ADAGRAD ERROR RATE

Why does ADAGRAD not deliver same/equal error rate compared to CW?

Binary features => square of 1 = 1

Class imbalance => it does affect the gradients

Very sparse feature vectors (short email messages) => make weight assignment based on gradient using small number of data points

Sparsity => Perhaps we can back off on the regularization to trade some sparsity for a better error rate
WELL BALANCED MACHINES

Our challenges:
Feature space grows over time
Content changes over time
Tag space grows over time
Tag space changes over time
Laptop environment is resource constrained

ADAGRAD latency/efficiency is a huge improvement over CW
ADAGRAD sparsity is a win
ADAGRAD error rate needs work

Next steps: Figure out how to fix error rate and test it in the real world! 😊