Unconstrained Optimization
Reminders

• Homework C: Data Structures
  – Out: Mon, Nov. 26
  – Due: Mon, Dec. 3 at 11:59pm

• Quiz B: Computation; Programming & Efficiency
  – Wed, Dec. 5, in-class
  – Covers Lectures 7 – 12

• Homework D: Inference & Optimization
  – Out: Mon, Dec. 3
  – Due: Fri, Dec. 7 at 11:59pm
Q&A
GRADIENT DESCENT
Motivation: Gradient Descent

Cases to consider gradient descent:
1. What if we **can not** find a closed-form solution?
2. What if we **can**, but it’s inefficient to compute?
3. What if we **can**, but it’s numerically unstable to compute?
Motivation: Gradient Descent

To solve the Ordinary Least Squares problem we compute:

$$\hat{\theta} = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (y^{(i)} - (\theta^T x^{(i)}))^2$$

$$= (X^T X)^{-1} (X^T Y)$$

The resulting shape of the matrices:

$$\left( \begin{array}{ccc} X^T & X \end{array} \right)_{M \times N} \left( \begin{array}{c} X \end{array} \right)_{N \times M}^{-1} \left( \begin{array}{c} X^T \ Y \end{array} \right)_{M \times N}$$

Background: Matrix Multiplication

- If $A$ is $q \times r$ and $B$ is $r \times s$, computing $AB$ takes $O(qrs)$
- If $A$ and $B$ are $q \times q$, computing $AB$ takes $O(q^{2.373})$
- If $A$ is $q \times q$, computing $A^{-1}$ takes $O(q^{2.373})$.

Computational Complexity of OLS:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X^T X$</td>
<td>$O(M^2 N)$</td>
</tr>
<tr>
<td>$(X^T X)^{-1}$</td>
<td>$O(M^{2.373})$</td>
</tr>
<tr>
<td>$X^T Y$</td>
<td>$O(MN)$</td>
</tr>
<tr>
<td>$(X^T Y)^{-1}$</td>
<td>$O(M^2)$</td>
</tr>
<tr>
<td>total</td>
<td>$O(M^2 N + M^{2.373})$</td>
</tr>
</tbody>
</table>
Regularized Regret

Chalkboard

– Regularized loss minimization
– L1 vs. L2 regularizers
– Learning linear models by optimization
– Example loss functions:
  • zero-one
  • logistic (log-loss)
  • exponential
  • squared
  • hinge
  • perceptron
Losses for Linear Models

Figure from https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_loss_functions.html
Topographical Maps
Topographical Maps
These are the **gradients** that Gradient **Ascent** would follow.
These are the negative gradients that Gradient Descent would follow.
Shown are the paths that Gradient Descent would follow if it were making infinitesimally small steps.
Pros and cons of gradient descent

• Simple and often quite effective on ML tasks
• Often very scalable
• Only applies to smooth functions (differentiable)
• Might find a local minimum, rather than a global one
Gradient Descent

Chalkboard
- Gradient Descent Algorithm
- Details: starting point, stopping criterion, line search
Algorithm 1 Gradient Descent

1: procedure GD($\mathcal{D}$, $\theta^{(0)}$)
2: $\theta \leftarrow \theta^{(0)}$
3: while not converged do
4: $\theta \leftarrow \theta - \lambda \nabla_{\theta} J(\theta)$
5: return $\theta$

In order to apply GD to Linear Regression all we need is the gradient of the objective function (i.e. vector of partial derivatives).

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \frac{d}{d\theta_1} J(\theta) \\ \frac{d}{d\theta_2} J(\theta) \\ \vdots \\ \frac{d}{d\theta_M} J(\theta) \end{bmatrix}$$
Gradient Descent

Algorithm 1 Gradient Descent

1: procedure GD(D, \( \theta^{(0)} \))
2: \( \theta \leftarrow \theta^{(0)} \)
3: while not converged do
4: \( \theta \leftarrow \theta - \lambda \nabla_{\theta} J(\theta) \)
5: return \( \theta \)

There are many possible ways to detect convergence. For example, we could check whether the L2 norm of the gradient is below some small tolerance.

\[ \| \nabla_{\theta} J(\theta) \|_2 \leq \epsilon \]

Alternatively we could check that the reduction in the objective function from one iteration to the next is small.
STOCHASTIC GRADIENT DESCENT
Stochastic Gradient Descent (SGD)

**Algorithm 2 Stochastic Gradient Descent (SGD)**

1: procedure SGD($\mathcal{D}, \theta^{(0)}$)
2: $\theta \leftarrow \theta^{(0)}$
3: while not converged do
4:     for $i \in$ shuffle({1, 2, \ldots, $N$}) do
5:         $\theta \leftarrow \theta - \lambda \nabla_{\theta} J^{(i)}(\theta)$
6: return $\theta$

We need a per-example objective:

\[
J(\theta) = \sum_{i=1}^{N} J^{(i)}(\theta)
\]
Convergence Curves

- **SGD** reduces **MSE** much more rapidly than **GD**
- **For GD / SGD**, training **MSE** is initially large due to uninformed initialization

---

**Figure adapted from Eric P. Xing**

- **Def**: an **epoch** is a single pass through the training data
  1. **For GD**, only **one update** per epoch
  2. **For SGD**, **N updates** per epoch
     \[ N = (\# \text{train examples}) \]

- **SGD** reduces **MSE** much more rapidly than **GD**
- **For GD / SGD**, training **MSE** is initially large due to uninformed initialization
Expectations of Gradients

\[
\frac{dJ(\theta)}{d\theta_j} = \frac{1}{N} \sum_{i=1}^{N} \frac{d}{d\theta_j} (J_i(\theta))
\]

\[
\nabla J(\theta) = \left[ \begin{array}{c} \vdots \\ \vdots \\ \nabla_j J(\theta) \\ \vdots \\ \vdots \end{array} \right] = \frac{1}{N} N_{i=1}^{N} \nabla J_i(\theta)
\]

Recall: for any discrete r.v. \( X \)

\[
\mathbb{E}_X \sum_{x} f(x) I \triangleq \sum_{x} P(x=x) f(x)
\]

Q: What is the expected value of a randomly chosen \( \nabla J_i(\theta) \)?

Let \( I \sim \text{Uniform}(\xi_1, \ldots, N^3) \)

\[
\Rightarrow P(I=i) = \frac{1}{N} \text{ if } i \in \{1, \ldots, N^3\}
\]

\[
\mathbb{E}_I [\nabla J_I(\theta)] = \sum_{i=1}^{N^3} P(I=i) \nabla J_i(\theta)
\]

\[
= \frac{1}{N} \sum_{i=1}^{N^3} \nabla J_i(\theta)
\]

\[
= \nabla J(\theta)
\]
## Convergence of Optimizers

### Convergence Analysis:

**Def:** Convergence is when $J(\hat{\theta}) - J(\theta^*) < \epsilon$

<table>
<thead>
<tr>
<th>Methods</th>
<th>Steps to Converge</th>
<th>Computation per iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton's Method</td>
<td>$O(\ln \ln 1/\epsilon)$</td>
<td>$\nabla J(\theta)$, $\nabla^2 J(\theta) \sim O(NM^2)$</td>
</tr>
<tr>
<td>GD</td>
<td>$O(\ln 1/\epsilon)$</td>
<td>$\nabla J(\theta) \sim O(NM)$</td>
</tr>
<tr>
<td>SGD</td>
<td>$O(1/\epsilon)$</td>
<td>$\nabla J(\theta) \sim O(M)$</td>
</tr>
</tbody>
</table>

"Almost sure" convergence with counts and conditions.

### Takeaway:
- SGD has much slower asymptotic convergence.
- But is often faster in practice.
ADAGRAD
Comparison of Algorithms

Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html
Comparison of Algorithms

Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html
Comparison of Algorithms

Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html
Comparison of Algorithms

Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html
Online Learning Algorithms

Algorithm Updates:

SGD: \[ w_{t+1} = w_t - \eta_t (f'_t(w_t) + r'(w_t)) \]

MD: \[ w_{t+1} = \arg \min_{w \in \Omega} \eta \langle f'_t(w_t), w - w_t \rangle + \eta r(w) + B_\psi(w, w_t) \]

COMID: \[ w_{t+1} = \arg \min_{w \in \Omega} \eta \langle f'_t(w_t), w - w_t \rangle + \eta r(w) + B_\psi(w, w_t) \]

RDA: \[ w_{t+1} = \arg \min_{w \in \Omega} \eta \langle \bar{g}_t, w \rangle + \eta r(w) + \frac{1}{t} \psi_t(w) \]

AdaGrad-COMID: \[ w_{t+1} = \arg \min_{w \in \Omega} \eta \langle f'_t(w_t) - H_tw_t, w \rangle + \eta r(w) + \frac{1}{2} \langle w, H_tw \rangle \]

AdaGrad-RDA: \[ w_{t+1} = \arg \min_{w \in \Omega} \eta \langle t\bar{g}_t, w \rangle + \eta r(w) + \frac{1}{2} \langle w, H_tw \rangle \]
Online Learning Algorithms

Derived Algorithms:

\( \ell_1 \)-regularization: For the regularizer \( r(w) = \lambda ||w||_1 \), we have the following updates.

- **RDA:** \( w_{t+1,i} = \text{sign}(-\bar{g}_{t,i})\eta\sqrt{t}[|\bar{g}_{t,i}| - \lambda]_+ \)

- **AdaGrad-RDA:** \( w_{t+1,i} = \text{sign}(-\bar{g}_{t,i})\frac{\eta t}{H_{t,ii}}[|\bar{g}_{t,i}| - \lambda]_+ \)

- **Fobos (COMID):** \( w_{t+1,i} = \text{sign}(w_{t,i} - \eta g_{t,i} [|w_{t,i} - \eta g_{t,i}| - \eta \lambda]_+ \)

- **AdaGrad-COMID:** \( w_{t+1,i} = \text{sign}\left( w_{t,i} - \frac{\eta}{H_{t,ii}} g_{t,i} \right) \left[ |w_{t,i} - \frac{\eta}{H_{t,ii}} g_{t,i}| - \frac{\lambda \eta}{H_{t,ii}} \right]_+ \)

where \([x]_+ = \max(0, x)\).

\( g_{t,i} \) is shorthand for the \( i \)th element of \( f_t'(\theta) \).
Derived Algorithms:

AdaGrad-COMID

For the $\ell_2^2$-regularizer, $r(\theta) = \frac{1}{2}||\theta||_2^2$, we have the following update.

$$\theta_i^{(t+1)} = \frac{H_{t,i,i} \theta_i^{(t)} - \eta g_{t,i}}{\eta \lambda \delta + H_{t,i,i}}$$

where the hyperparameter $\delta$ helps deal with the initially noisy values in $H_{t,i,i}$ and typically takes a small positive value $\leq 1$.

$H_t$ is a diagonal matrix defined such that each

$$H_{t,i,i} = \delta + \sqrt{\sum_{s=1}^{t} (f_s'(\theta)_i)^2}$$

is a smoothed version of the square root of the sum of the squares of the $i$th element of
Online Learning Algorithms

Derived Algorithms:

AdaGrad-COMID

In the case of no regularizer (i.e. \( r(\theta) = 0 \)), we have the following update.

\[
\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{H_{t,i,i} + \delta}} g_{t,i}
\]

\( H_t \) is a diagonal matrix defined such that each

\[
H_{t,i,i} = \delta + \sqrt{\sum_{s=1}^{t} (f'_s(\theta)_i)^2}
\]

is a smoothed version of the square root of the sum of the squares of the \( i \)th element of