COMP 330: Optimization–Grad Descent

Chris Jermaine and Kia Teymourian

Rice University

Optimization

At the heart of all "learning" frameworks discussed

- Is optimization!

Why?

- Well, it's explicit in case of loss funcs, MLE
- Implicitly in case of Bayesian

Means we need to ask: how to solve optimization problems?

- Fundamental question in data science!!

Desired Properties

To be useful for data science, opt framework should be

- Easily applied to many types of opt problems
- Scalable (easily built in MapReduce, for example)
- Fast (quick convergence)

Most Widely Used Opt Framework Is...

For (big) data science, at least...

- Gradient descent!

What's the idea?

- GD is an iterative algorithm
- Goal: choose Θ^* to min $L(\Theta)$
- Tries to incrementally improve current solution
- At step i, Θ_i is current guess for Θ^*

Gradient Descent

Basic algorithm:

$\Theta_1 \leftarrow$ non-stupid guess for Θ^*;
$i \leftarrow 1$;
repeat {
  $\Theta_{i+1} \leftarrow \Theta_i - \lambda \nabla L(\Theta_i)$;
  $i \leftarrow i + 1$;
} while $(||\Theta_i - \Theta_{i-1}||_1 < \epsilon)$

Gradient Descent

Basic algorithm:

$\Theta_1 \leftarrow$ non-stupid guess for Θ^*;
$i \leftarrow 1$;
repeat {
  $\Theta_{i+1} \leftarrow \Theta_i - \lambda \nabla L(\Theta_i)$;
  $i \leftarrow i + 1$;
} while $(||\Theta_i - \Theta_{i-1}||_1 < \epsilon)$

- Here λ is the "learning rate"
- Controls speed of convergence
- And $\nabla L(\Theta_i)$ is the gradient of L evaled at Θ_i
- What's a "gradient"?

A Gradient

Gradient is the multi-dimensional analog to a derivative

- If $L(.)$ accepts a vector
- ∇L is a vector-valued function
- That is, accepts a vector Θ
- Returns a vector...
- whose ith entry is ith partial derivative evaluated at Θ

Example

Returning to linear regression...

- Want a line to fit points $\langle 118, 122, 145, 149, 186 \rangle$
- At time ticks t in $\langle 1, 2, 3, 4, 5 \rangle$
- Prediction $f(t|c, m) = c + m \times t$
- Loss $L(c, m) = \sum_i (f(t_i|c, m) - x_i)^2$

Example

Returning to linear regression...

- Want a line to fit points $\langle 118, 122, 145, 149, 186 \rangle$
- At time ticks t in $\langle 1, 2, 3, 4, 5 \rangle$
- Prediction $f(t|c, m) = c + m \times t$
- Loss $L(c, m) = \sum_i (f(t_i|c, m) - x_i)^2$

Example

- Prediction $f(t|c,m) = c + m \times t$
- Loss $L(c,m) = \sum_i \left(f(t_i|c,m) - x_i\right)^2$

First we deal with c and m:

$$\frac{\partial F}{\partial c} = \sum_i 2(f(t_i|c,m) - x_i)$$

$$\frac{\partial F}{\partial m} = \sum_i 2t_i(f(t_i|c,m) - x_i)$$

So $\nabla L(c,m) =$

$$\langle \sum_i 2(f(t_i|c,m) - x_i), \sum_i 2t_i(f(t_i|c,m) - x_i) \rangle$$

Grad Descent and Big Data

So $\nabla L(c, m) =$

$$\langle \sum_i 2(f(t_i|c, m) - x_i), \sum_i 2t_i(f(t_i|c, m) - x_i) \rangle$$

Gradient of this form is very common

Why is this so good for "big data", MapReduce?

The Learning Rate

Reconsider the algorithm:

$\Theta_1 \leftarrow$ non-stupid guess for Θ^*;
$i \leftarrow 1$;
repeat {
  $\Theta_{i+1} \leftarrow \Theta_i - \lambda \nabla L(\Theta_i)$;
  $i \leftarrow i + 1$;
} while $(||\Theta_i - \Theta_{i-1}||_1 < \epsilon)$

How to choose λ?

Super important

- Too small: many, many passes thru the data to converge
- Too large: oscillate into oblivion

Line Search

Best option (in terms of results) but most expensive:

- Solve another mini-optimization problem
- That is, choose λ so as to minimize $L(\Theta_{i+1})$
- At lest now, it's a 1-dimensional opt problem!
- Called a "line search"

Line Search

$l \leftarrow 0$;
$h \leftarrow 999999$;
while $(h - l > \epsilon)$ do {
  $h' \leftarrow l + \frac{1}{c}(h - l)$;
  $l' \leftarrow h - \frac{1}{c}(h - l)$;
  $\text{goodness}_h \leftarrow L(\Theta_i - h'\nabla L(\Theta_i))$;
  $\text{goodness}_l \leftarrow L(\Theta_i - l'\nabla L(\Theta_i))$;
  if (goodness_h < goodness_l) {
    $l \leftarrow l'$;
  else
    $h \rightarrow h'$;
  }
}

"Golden section search": $c = \frac{1}{2}(1 + \sqrt{5}) = 1.618$

Other Ways To Choose Learning Rate

Line search is costly!

Other standard method is "Bold Driver"

- Make a very conservative initial guess for λ
- At each iteration, compute $L(\Theta_i)$
- Better than last time? $\lambda \leftarrow \lambda \times 1.05$
- Worse than last time? $\lambda \leftarrow \lambda \times 0.5$
- Just one eval of loss function per iteration!

Questions?