

Tools & Models for Data Science

Optimization—Gradient Descent

Chris Jermaine & Risa Myers

Rice University



- At the heart of all “learning” frameworks discussed
 - Is optimization!
- Why?
 - Well, it's explicit in case of loss functions, MLE
 - Implicitly in case of Bayesian
- Means we need to ask: how to solve optimization problems?
 - Fundamental question in data science!!

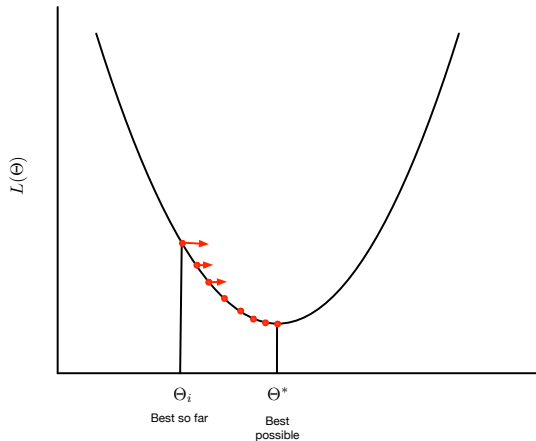
- To be useful for data science, an optimization framework should be
 - Easily applied to many types of optimization problems
 - Scalable (easily built in Spark, for example)
 - Fast (quick convergence)

Most Widely Used Optimization Framework Is...

- For (big) data science, at least...
 - Gradient descent!
- What's the idea?
 - GD is an iterative algorithm
 - Goal: choose Θ^* to minimize/maximize the Loss function $L(\Theta)$
 - Tries to incrementally improve current solution
 - At step i , Θ_i is current guess for Θ^*

Gradient Descent Intuition

- Look at the slope of L
- Go in the direction of steepest descent
- Don't go too far!
- Stop when your parameter values aren't changing much
- Can end up oscillating if your step size is too big



■ Basic algorithm:

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

- $\lambda \nabla L(\Theta_i)$ is some distance along the direction of steepest descent
- $\|\Theta_i - \Theta_{i-1}\|_1$ is the L₁ distance, the absolute value of the change in parameter value(s)

■ Basic algorithm:

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

■ λ is the “learning rate”

- how far you go a each step in the direction of steepest descent
- Controls speed of convergence
- If too big, algorithm will oscillate
- If too small, algorithm will take a very long time to run

■ $\nabla L(\Theta_i)$ is the gradient of L evaluated at Θ_i

Stopping Condition

- Here we use

```
while ( $\|\Theta_i - \Theta_{i-1}\|_1 > \epsilon$ )
```

- Easy to compute
- Efficient because it just requires checking for small changes in **model**
- Serves as a **proxy** for the change in the **loss function**
- But does not always make sense (big change in model can mean small change in accuracy)

Stopping Condition

- If feasible, instead use

`while (| $L(\Theta_i) - L(\Theta_{i-1})$ | > ϵ)`

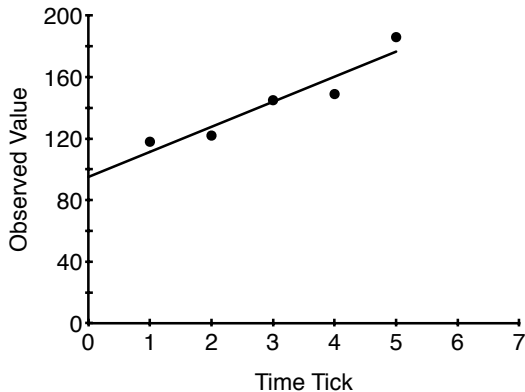
- Drawback: requires loss computation... can be expensive
- Can be more expensive than another iteration
- Alternative: Stochastic Gradient Descent or Minibatch
 - Use a small sample of the dataset
 - Note: the parameters (Θ) will never stop changing, due to different data points used each time
 - Much more difficult to decide when to stop

- What's a “gradient”?
- Gradient is the multi-dimensional analog to a derivative
 - If $L(\cdot)$ accepts a vector
 - ∇L is a vector-valued function
 - That is, accepts a vector Θ
 - Returns a vector, Θ'
 - whose i th entry is i th 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 $f(t|b, m) = b + m \times t$
- Loss $L(b, m) = \sum_i (f(t_i|b, m) - x_i)^2$
- Model parameters: $\{b, m\}$
{intercept, slope}
- Use L_2 loss: Least Squares



- So $\nabla L(b, m) =$

$$\left\langle \sum_i 2(f(t_i|b, m) - x_i), \sum_i 2t_i(f(t_i|b, m) - x_i) \right\rangle$$

- Derivation is left as an exercise to the user (or look at the last slide in the deck)
- Gradient of this form (summing up values over all the data points) is very common
- ? Why is this so good for “big data”, MapReduce/Spark?

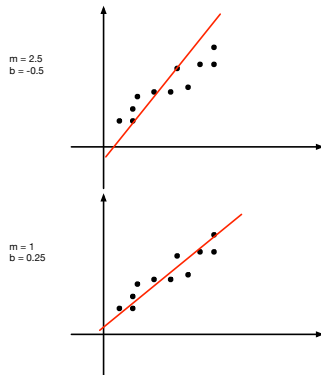
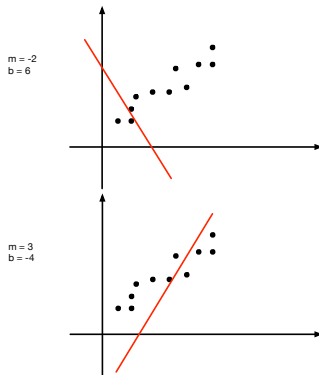
- So $\nabla L(b, m) =$

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

- Gradient of this form (summing up values over all the data points) is very common
- Why is this so good for “big data”, MapReduce/Spark?
 - Sums are easily parallelized

What do Gradient Descent Steps Look Like?

```
 $\Theta_1 \leftarrow \text{non-stupid guess for } \Theta^*;$   
repeat {  
  update m  
  update b  
} while ( $\|\Theta_i - \Theta_{i-1}\|_1 > \epsilon$ )
```



The Learning Rate

- How much do we change the parameter values each iteration?
- Reconsider the algorithm:

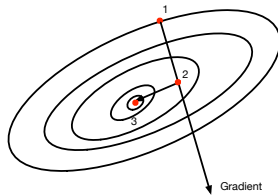
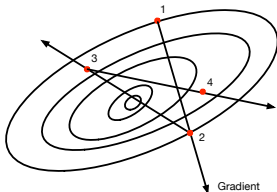
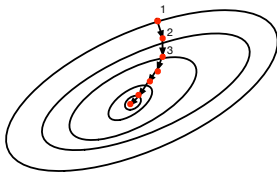
```
 $\Theta_1 \leftarrow$  non-stupid guess for  $\Theta^*$ ;  
 $i \leftarrow 1$ ;  
repeat {  
   $\Theta_{i+1} \leftarrow \Theta_i - \lambda \nabla L(\Theta_i)$ ;  
   $i \leftarrow i + 1$ ;  
} while ( $\|\Theta_i - \Theta_{i-1}\|_1 > \varepsilon$ )
```

- How to choose λ ?

Choosing the Learning Rate: λ

- Super important

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



Choosing the Learning Rate

- There are two classic approaches
 - 1 Line search
 - 2 Bold driver

- Best option (in terms of results) but most expensive:
 - Solve another mini-optimization problem at each iteration
 - That is, choose λ so as to minimize $L(\Theta_{i+1})$
 - At least now, it's a 1-dimensional optimization problem!
 - Called a “line search”

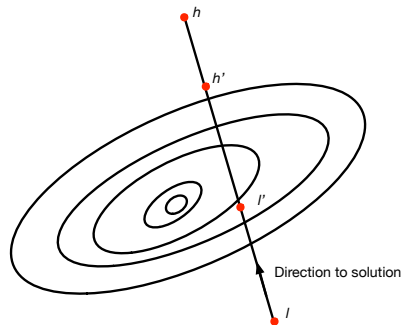
- Sort of like a binary search
- But try to find a minimum, not a specific value
- What is binary search?

Binary Search

- Also called half-interval search
- Finds the position of a value within a sorted array
- Say you are trying to load data into a database
- But the load fails and gives no error
- You want to figure out where it is failing
- You might try to load the first half of the data
- If that works, the error must be in the 2nd half
- So, try to load the first half of that data
- ...
- Average performance is $O(\log n)$

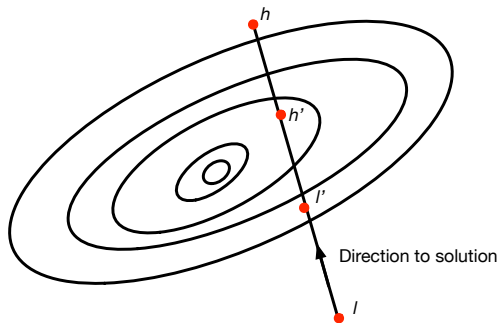
Line Search

- Sort of like a binary search
- But try to find a minimum, not a specific value
 - 1 Find the direction that minimizes the objective function
 - 2 Decide how big of a step to take in that direction
- Always have two extreme bounds l and h on λ
- At each iteration, choose two l' , h' within $[l, h]$
- Breaks line segment between l and h three ways (two ends and a middle)
- Evaluate loss at l' , h'
- Cut off the worse of the two ends



Line Search

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



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

Problems with Line Search

- Line search is costly!
 - We have to keep recalculating the value of the loss function
 - This is not feasible for big data!

Other Ways To Choose Learning Rate

- Other standard method is “Bold Driver”
 - Widely used approach
- Approach
 - Make a very conservative initial guess for λ
 - At each iteration, compute $L(\Theta_i)$
 - Better than last time? Increment λ just a little bit: $\lambda \leftarrow \lambda \times 1.05$
 - Worse than last time? Reduce λ by a lot: $\lambda \leftarrow \lambda \times 0.5$
 - Just one eval of loss function per iteration!

- Increase λ slowly so we don't miss a divergence
- At first sign of a divergence, back up massively
- Theory says that if you choose reasonable increment and decrement factors, the algorithm is guaranteed to converge

Questions?

? What do we know now that we didn't know before?

? How can we use what we learned today?

Questions?

- What do we know now that we didn't know before?
 - How to solve optimization problems using Gradient Descent
 - Different ways to decide to terminate learning
 - Some issues with Big Data
 - Some approaches for adapting the learning rate in Gradient Descent
- How can we use what we learned today?
 - In the next exercise!

Example

- Prediction $f(t|b, m) = b + m \times t$
- Loss $L(b, m) = \sum_i (f(t_i|b, m) - x_i)^2$
- First we deal with b and then with m :

$$\begin{aligned}\frac{\partial L}{\partial b} &= \frac{\partial \sum_i (f(t_i|b, m) - x_i)^2}{\partial b} \\&= \sum_i 2(f(t_i|b, m) - x_i) \frac{\partial (f(t_i|b, m) - x_i)}{\partial b} \\&= \sum_i 2(f(t_i|b, m) - x_i) \\\frac{\partial L}{\partial m} &= \frac{\partial \sum_i (f(t_i|b, m) - x_i)^2}{\partial m} \\&= \sum_i 2(f(t_i|b, m) - x_i) \frac{\partial (f(t_i|b, m) - x_i)}{\partial m} \\&= \sum_i 2t_i(f(t_i|b, m) - x_i)\end{aligned}$$