

Tools & Models for Data Science

Optimization—Newton's Method

Chris Jermaine & Risa Myers

Rice University



Gradient Descent

- 1 Goal: Find the parameter values for a model that best fit our data
- 2 Find the best direction to go in
- 3 Take a [sized] step in that best direction
- 4 Repeat until convergence

Alternatives to Gradient Descent

- Gradient descent is great
 - Easy to use
 - Widely applicable
 - But convergence can be slow
- Can we do better? Sure!

- Class of iterative optimization methods
 - Use not only first partial derivatives
 - But second as well
 - Speeds convergence
 - Cost: more complexity
 - Cost: quadratic in number of variables

- Classic second order method for optimization¹
- Comes from Newton's method for finding zero of a function $F()$
- Recall that the “zeroes” of a function are the roots / solutions
- Use this approach to keep finding better approximations to the zeros of a function

¹second order method means it uses the second derivative

- 1 Review of Newton's method for finding the root of a **1 variable function**
- 2 Introduce method for finding the root of a **1 variable gradient of a Loss function**
- 3 Review of Newton's method for finding the root of a **multi-variable function**
- 4 Introduce method for finding the root of a **multi-variable gradient of a Loss function**

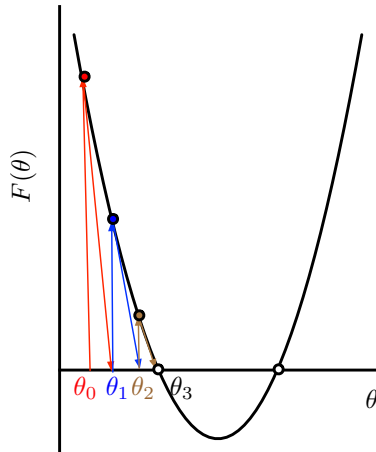
```
 $\theta \leftarrow \text{intial guess};$   
while  $\theta$  keeps changing, do:  
     $\theta \leftarrow \theta - F'(\theta)^{-1}F(\theta);$ 
```

- θ is the value of the model parameter

- Make an initial guess
- Approximate $F(\theta)$ with a line (tangent to $F()$ at that guess)
- Update θ

Newton's Method Intuition

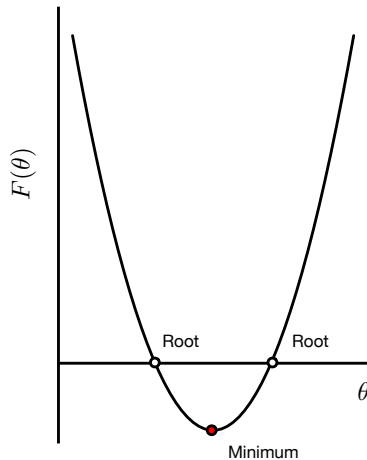
- 1 Pick a value for θ
- 2 Evaluate $F(\theta)$
- 3 Evaluate the derivative of the function at θ , $F'(\theta)$
- 4 Revise θ based on these values
- 5 Repeat until convergence of θ



Isn't that what we did in Gradient Descent?

Key difference:

- Root finding, not minimum finding
- Want the zero of the **function**
- Not the zero of the **derivative**



Newton's Method for Optimization

- In data science, don't want a zero
 - We want a max/min of loss function $L()$
 - So, just find the root (zero) of the derivative $L'()$
 - So, $F(\theta) \rightarrow L'(\theta)$
- Algorithm becomes:

```
 $\theta \leftarrow$  initial guess;  
while  $\theta$  keeps changing, do:  
   $\theta \leftarrow \theta - \frac{L'(\theta)}{L''(\theta)};$ 
```

Multi-Variate Newton's Method

- Say we have a multi-variate function $F : \mathbb{R}^d \rightarrow \mathbb{R}^d$, where d is the number of dimensions
 - The i th output of F is given by the function F_i
 - So $F(\Theta) = \langle F_1(\Theta), F_2(\Theta), \dots, F_d(\Theta) \rangle$
- We want to find a zero of F ; that is, find $\Theta = \langle \theta_1, \theta_2, \dots, \theta_d \rangle$ such that:

$$F_1(\theta_1, \theta_2, \dots, \theta_d) = 0$$

$$F_2(\theta_1, \theta_2, \dots, \theta_d) = 0$$

...

$$F_d(\theta_1, \theta_2, \dots, \theta_d) = 0$$

- How to do this?

Multi-Variate Newton's Method

- Turns out it's not so difficult...
 - Won't do the derivation (relies on multi-variate Taylor expansion)
 - From Linear Algebra: a Jacobian Matrix contains all the partial first derivatives of a function
 - Here, F_i is the function that governs the i th dimension
 - Define the "Jacobian" of F to be:

$$J_F = \begin{pmatrix} \frac{\partial F_1}{\partial \theta_1} & \frac{\partial F_1}{\partial \theta_2} & \frac{\partial F_1}{\partial \theta_3} & \cdots \\ \frac{\partial F_2}{\partial \theta_1} & \frac{\partial F_2}{\partial \theta_2} & \frac{\partial F_2}{\partial \theta_3} & \cdots \\ \frac{\partial F_3}{\partial \theta_1} & \frac{\partial F_3}{\partial \theta_2} & \frac{\partial F_3}{\partial \theta_3} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

- Note: this is a $d \times d$ matrix of functions!
- Rows cover the different parameters for a single dimension
- Columns cover the value of F for each dimension for a single parameter
- We can evaluate it at any set of parameter values
- So $J_F(\Theta)$ is a matrix of scalars

- Multi-Variate Newton's is simply:

```
 $\Theta \leftarrow \text{intial guess};$   
while  $\Theta$  keeps changing, do:  
     $\Theta \leftarrow \Theta - J_F^{-1}(\Theta)F(\Theta);$ 
```

- Update each model parameter using the inverse of the Jacobian evaluated at the current values of the model parameters, Θ , and the value of the function using the same parameters

What About Multi-Variate Optimization?

- Again, we want to solve an optimization problem, not find function roots
- Difference: we don't have a system of equations to solve
- In multidimensional space, this is equivalent to standing at the top of a mountain or bottom of a valley
 - Just have a loss function $L()$, which we want to minimize (or maximize)
 - Min/max is at Θ such that:

$$\frac{\partial L}{\partial \theta_1}(\Theta) = 0$$

$$\frac{\partial L}{\partial \theta_2}(\Theta) = 0$$

...

$$\frac{\partial L}{\partial \theta_d}(\Theta) = 0$$

- That is, we want Θ such that $\nabla L(\Theta) = \langle 0, 0, \dots, 0 \rangle$
- Can then use exactly the same algorithm as before [MV Newton's Method] to find root of $\nabla L(\Theta)$

To find max/min, then this:

```
 $\Theta \leftarrow \text{intial guess};$   
while  $\Theta$  keeps changing, do:  
     $\Theta \leftarrow \Theta - J_F^{-1}(\Theta)F(\Theta);$ 
```

Becomes this:

```
 $\Theta \leftarrow \text{intial guess};$   
while  $\Theta$  keeps changing, do:  
     $\Theta \leftarrow \Theta - J_{\nabla L}^{-1}(\Theta)\nabla L(\Theta);$ 
```

- We are taking the Jacobian over the gradient instead of over a system of equations, F_i

One Last Thing

We have:

```
 $\Theta \leftarrow \text{intial guess};$   
while  $\Theta$  keeps changing, do:  
     $\Theta \leftarrow \Theta - J_{\nabla L}^{-1}(\Theta) \nabla L(\Theta);$ 
```

- The matrix of functions $J_{\nabla L}$ is typically called the “Hessian” of L
- Entries are:

$$H_L = \begin{pmatrix} \frac{\partial L}{\partial \theta_1^2} & \frac{\partial L}{\partial \theta_1 \partial \theta_2} & \frac{\partial L}{\partial \theta_1 \partial \theta_3} & \cdots \\ \frac{\partial L}{\partial \theta_1 \partial \theta_2} & \frac{\partial L}{\partial \theta_2^2} & \frac{\partial L}{\partial \theta_2 \partial \theta_3} & \cdots \\ \frac{\partial L}{\partial \theta_1 \partial \theta_3} & \frac{\partial L}{\partial \theta_2 \partial \theta_3} & \frac{\partial L}{\partial \theta_3^2} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

- Each entry is the 2nd derivative of the loss function with respect to each parameter
- Each row is the Jacobian given a set of model parameters, Θ

Pros and Cons of Newton's

- Pro: Convergence is quadratic; that is, error decreases quadratically
- Pro: Hundreds/thousands of iterations (gradient descent) becomes tens
- Pro: No learning rate to set
- Pro: Doesn't require $F(\Theta)$ to be convex

- Con: More complicated than gradient descent!
- Con: quadratic cost each iteration (linear gradient descent)
The Hessian is quadratic in the number of variables
- Actually, the cost is worse than quadratic, since the matrix has to be inverted
- Con: The second derivative has to exist

- Not used much in practice since in high dimensions, $d \times d$ is too big
- Usable for $< 100K$ parameters, really hard at 1M
- Quasi-Newton methods are used instead
- Typically use just a portion or estimation of the Hessian matrix
- E.g. Limited-memory BFGS

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?
 - We have another numerical method to find the parameters of a loss function
 - We understand how we can handle multi-dimensional data
- How can we use what we learned today?
 - We can find the model parameters using a different approach