UVA CS 6316 – Fall 2015 Graduate: Machine Learning

Lecture 4: More optimization for Linear Regression

Dr. Yanjun Qi

University of Virginia

Department of
Computer Science

Where we are ? ➔ Five major sections of this course

- ❑ Regression (supervised)
- ❑ Classification (supervised)
- ❑ Unsupervised models
- ❑ Learning theory
- ❑ Graphical models

Today →

Regression (supervised)

- ❑ Four ways to train / perform optimization for linear regression models
  - ❑ Normal Equation
  - ❑ Gradient Descent (GD)
  - ❑ Stochastic GD
  - ❑ Newton's method

- ❑ Supervised regression models
  - ❑ Linear regression (LR)
  - ❑ LR with non-linear basis functions
  - ❑ Locally weighted LR
  - ❑ LR with Regularizations

Today

- ❑ A Practical Application of Regression Model
- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD (SGD)
  - ❑ Newton's method

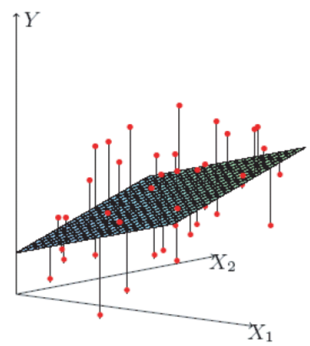
Linear Regression Models

$$\boxed{f} : X \longrightarrow Y$$

➔ e.g. Linear Regression Models

$$\hat{y} = f(x) = \theta_0 + \theta_1 x^1 + \theta_2 x^2$$

- Features:
  Living area, distance to campus, # bedroom …
- Target y:
  Rent ➔ Continuous



Dr. Yanjun Qi / UVA CS 6316 / f15

training / learning goal

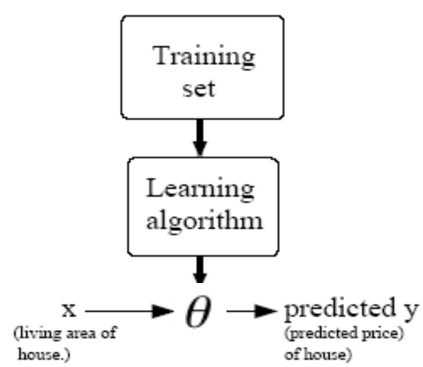
- Using matrix form, we get the following general representation of the linear function on train set:

$$\underbrace{\hat{\mathbf{Y}}}_{n\times 1} = \underbrace{X}_{n\times p}\underbrace{\theta}_{p\times 1}$$

- Our goal is to pick the optimal θ that minimize the following cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i(\vec{x}_i) - y_i)^2$$

Our goal:



Method I: normal equations

- Write the cost function in matrix form:

$$
\begin{aligned}
J(\theta) &= \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2 \\
&= \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\
&= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)
\end{aligned}
$$

$$
\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}
$$

To minimize $J(\theta)$, take its gradient and set to zero:

$$\Rightarrow \boxed{X^T X\theta = X^T\vec{y}}$$

The normal equations

$$\Downarrow$$

$$\theta^* = \left(X^T X\right)^{-1} X^T\vec{y}$$

e.g. A Practical Application of Regression Model

Movie Reviews and Revenues: An Experiment in Text Regression*

Mahesh Joshi Dipanjan Das Kevin Gimpel Noah A. Smith
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{maheshj,dipanjan,kgimpel,nasmith}@cs.cmu.edu

Abstract

We consider the problem of predicting a movie's opening weekend revenue. Previous work on this problem has used metadata about a movie—e.g., its genre, MPAA rating, and cast—with very limited work making use of text *about* the movie. In this paper, we use the text of film critics' reviews from several sources to predict opening weekend revenue. We describe a new dataset pairing movie reviews with metadata and revenue data, and show that review text can substitute for metadata, and even improve over it, for prediction.

Proceedings of HLT '2010 Human Language Technologies:

I. The Story in Short

- Use metadata and critics' reviews to predict opening weekend revenues of movies
- Feature analysis shows what aspects of reviews predict box office success

II. Data

- 1718 Movies, released 2005-2009
- Metadata (genre, rating, running time, actors, director, etc.): *www.metacritic.com*
- Critics' reviews (~7K): Austin Chronicle, Boston Globe, Entertainment Weekly, LA Times, NY Times, Variety, Village Voice
- Opening weekend revenues and number of opening screens: *www.the-numbers.com*

Movie Reviews and Revenues: An Experiment in Text Regression,
Proceedings of HLT '10 Human Language Technologies:

III. Model

- Linear regression with the elastic net (Zou and Hastie, 2005)

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}=(\beta_0,\boldsymbol{\beta})} \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - (\beta_0 + \boldsymbol{x}_i^\top \boldsymbol{\beta})\right)^2 + \lambda P(\boldsymbol{\beta})$$

$$P(\boldsymbol{\beta}) = \sum_{j=1}^{p}\left(\tfrac{1}{2}(1-\alpha)\beta_j^2 + \alpha|\beta_j|\right)$$

Use linear regression to directly predict the opening weekend gross earnings, denoted y, based on features x extracted from the movie metadata and/or the text of the reviews.

Movie Reviews and Revenues: An Experiment in Text Regression, Proceedings of HLT '10 Human Language Technologies:

IV. Features

I	Lexical n-grams (1,2,3)
II	Part-of-speech n-grams (1,2,3)
III	Dependency relations (nsubj,advmod,…)
Meta	U.S. origin, running time, budget (log), # of opening screens, genre, MPAA rating, holiday release (summer, Christmas, Memorial day,…), star power (Oscar winners, high-grossing actors)

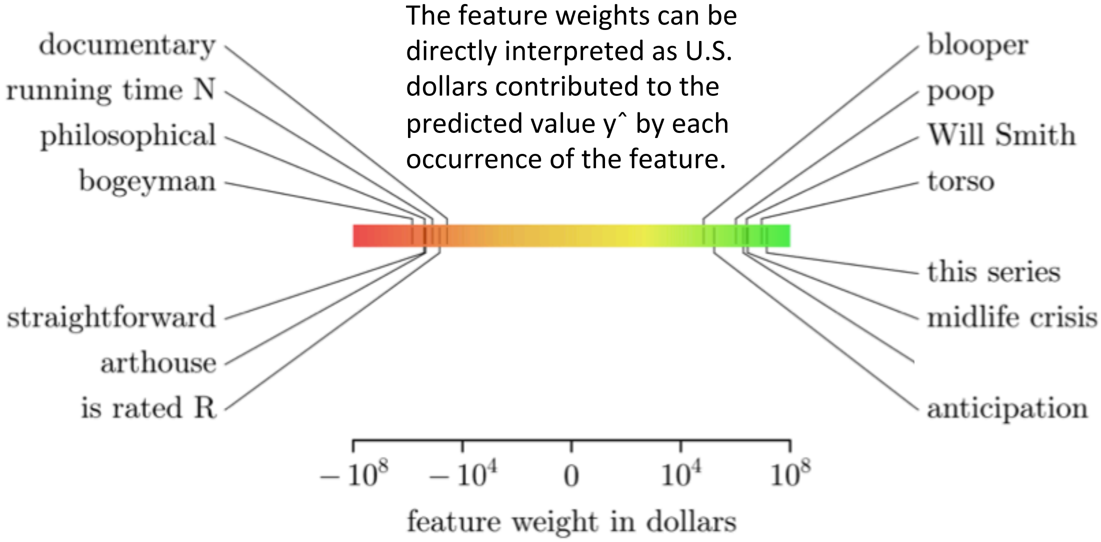
e.g. counts of a ngram in the text

VIII. Get the Data!

www.ark.cs.cmu.edu/movie$-data

V. What May Have Brought You to movies

The feature weights can be directly interpreted as U.S. dollars contributed to the predicted value y^ by each occurrence of the feature.



Today

- ❑ A Practical Application of Regression Model
- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient Descent
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD (SGD)
  - ❑ Newton's method

Review: Definitions of gradient (from CMU review handout)

Suppose that $f : \mathbb{R}^{m\times n} \rightarrow \mathbb{R}$ is a function that takes as input a matrix A of size $m \times n$ and returns a real value. Then the ***gradient*** of f (with respect to $A \in \mathbb{R}^{m\times n}$) is the matrix of

→ *Denominator layout*

$$\nabla_A f(A) \in \mathbb{R}^{m\times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

Review: Definitions of gradient (from http://en.wikipedia.org/wiki/Matrix_calculus#Scalar-by-matrix)

The derivative of a scalar *y* function of a matrix **X** of independent variables, with respect to the matrix **X** $_{p*q}$, is given as

→ *Numerator layout*

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{p1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{p2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{1q}} & \frac{\partial y}{\partial x_{2q}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix}$$

Notice that the indexing of the gradient with respect to **X** is transposed as compared with the indexing of **X**.

→ numerator layout

Review: Definitions of gradient (from CMU handout)

- Size of gradient is always the same as the size of

→ *Denominator layout*

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \quad \text{if } x \in \mathbb{R}^n$$

(from http://en.wikipedia.org/wiki/Matrix_calculus#Scalar-by-vector)

The derivative of a scalar y by a vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, is

→ *Numerator layout*

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

This gradient is a 1×n row vector whose entries respectively contain the n partial derivatives
→ numerator layout

A little bit more about [Optimization]

- Objective function $F(x)$
- Variables x
- Constraints

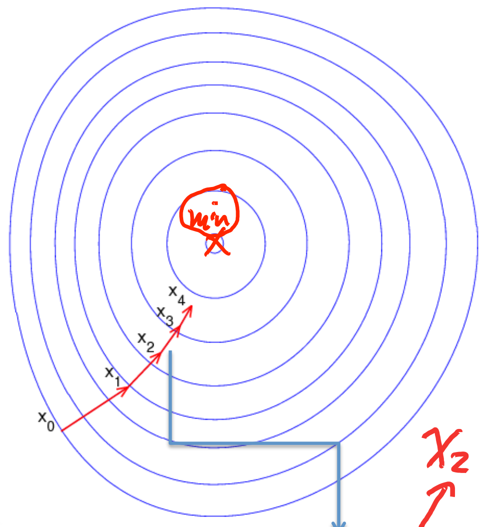
To find values of the variables
that minimize or maximize the objective function
while satisfying the constraints

e.g. Gradient Descent (Steepest Descent)

The gradient points in the direction of the greatest rate of increase of the function and its magnitude is the slope of the graph in that direction

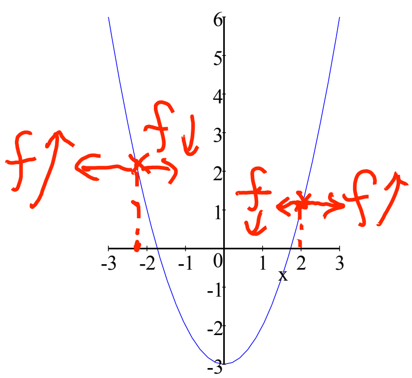
A first-order optimization algorithm.

To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient of the function at the current point.



$$-\nabla_x F(x_{k-1})$$

Review: Derivative of a Quadratic Function



$$y = x^2 - 3$$

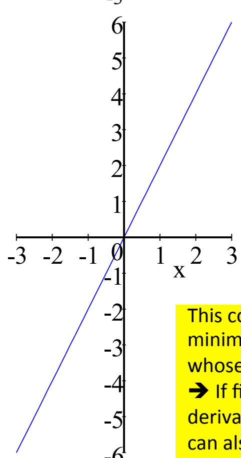
$$y' = \lim_{h \to 0} \frac{(x+h)^2 - 3 - (x^2 - 3)}{h}$$

$$y' = \lim_{h \to 0} \frac{\cancel{x^2} + 2x\cancel{h} + h^{\cancel{2}} - \cancel{x^2}}{\cancel{h}}$$

$$y' = \lim_{h \to 0} 2x + \overset{0}{\cancel{h}}$$

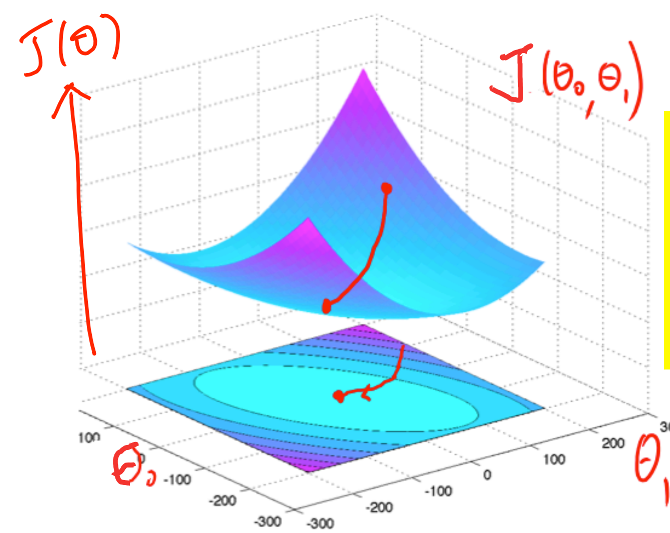
$$y' = 2x$$

$$y'' = 2$$



This convex function is minimized @ the unique point whose derivative (slope) is zero.
➔ If finding zeros of the derivative of this function, we can also find minima (or maxima) of that function.

Illustration of Gradient Descent (2D case)



The gradient points in the direction of the greatest rate of increase of the function and its magnitude is the slope of the graph in that direction

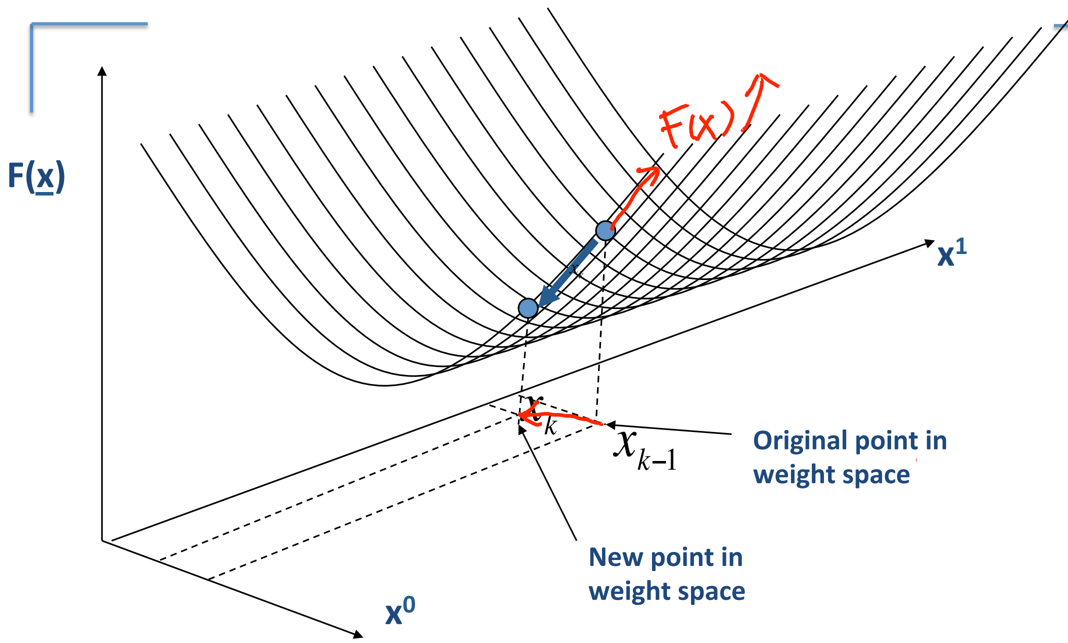
Gradient Descent (GD)

- Initialize k=0, choose x_0
- While $k<k_{max}$

For the k-th epoch

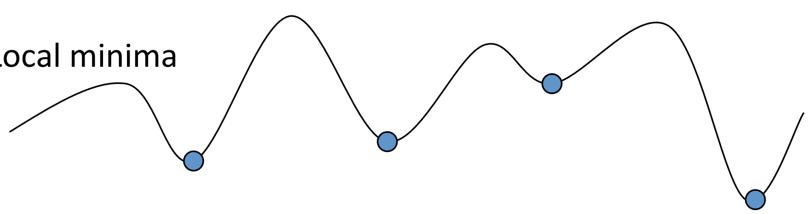
$$x_k = x_{k-1} - \alpha \nabla_x F(x_{k-1})$$

Illustration of Gradient Descent (2D case)



Comments on Gradient Descent Algorithm

- Works on any objective function F(x)
  - as long as we can evaluate the gradient
  - this can be very useful for minimizing complex functions
- Local minima



  - Can have multiple local minima
  - (note: for LR, its cost function only has a single global minimum, so this is not a problem)
  - If gradient descent goes to the closest local minimum:
    - solution: random restarts from multiple places in weight space

Today

- ❑ A Practical Application of Regression Model
- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient Descent
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD (SGD)
  - ❑ Newton's method

LR with batch GD

- The Cost Function:

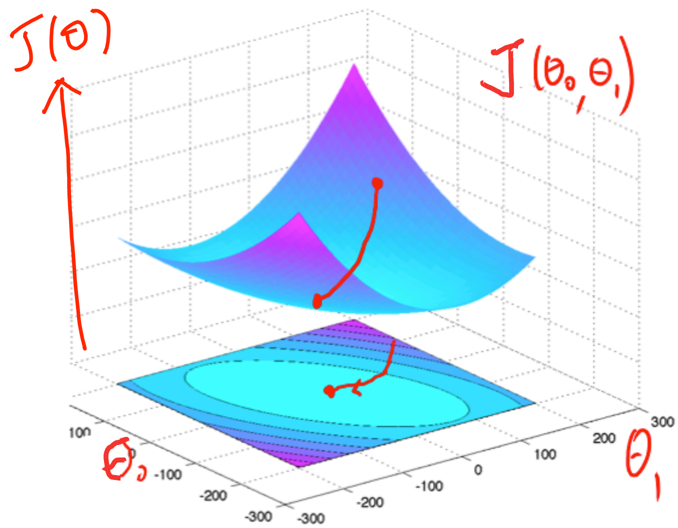
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$

- Consider a **gradient descent** algorithm:

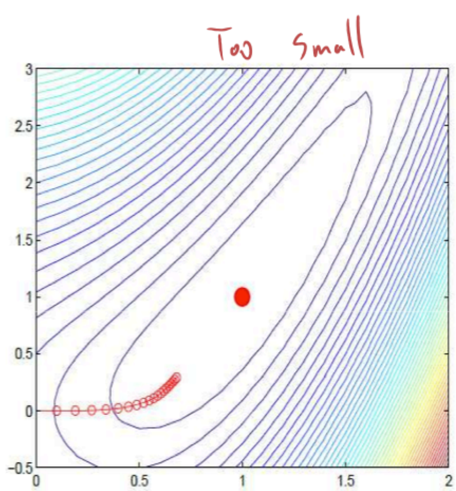
$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{p-1} \end{bmatrix} \qquad \theta_j^{t+1} = \theta_j^t - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\Big|_t$$

For the (t+1)-th epoch

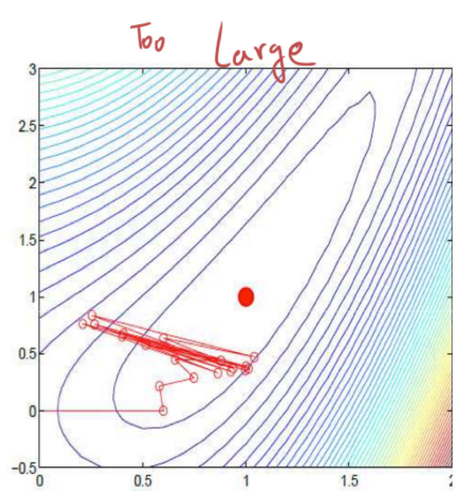
Illustration of Gradient Descent (2D case)



Choosing the Right Step-Size / Learning-Rate is critical



$\alpha = 0.1.$



$\alpha = 0.6.$

$$J(\theta) = (X\theta - y)^T (X\theta - y)\frac{1}{2}$$

$$= ((X\theta)^T - y^T)(X\theta - y)\frac{1}{2}$$

$$= (\theta^T X^T - y^T)(X\theta - y)\frac{1}{2}$$

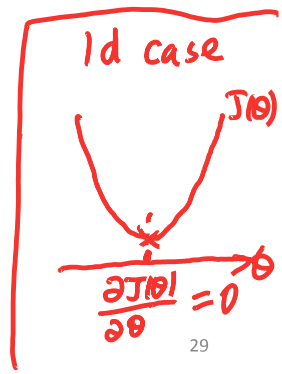
$$= (\theta^T X^T X\theta - \underbrace{\theta^T X^T y - y^T X\theta}_{} + y^T y)\frac{1}{2}$$

Since $\theta^T X^T y = y^T X\theta$

$\langle X\theta, y\rangle \quad \langle y, X\theta\rangle$

$$= (\theta^T X^T X\theta - 2\theta^T X^T y + y^T y)\frac{1}{2}$$

$\Rightarrow$ $J(\theta)$ quadratic func of θ;



See handout 4.1 + 4.3 $\Rightarrow$ matrix calculus, partial deri $\Rightarrow$ Gradient

$$\nabla_\theta(\theta^T X^T X\theta) = 2X^T X\theta \qquad (P24)$$

$$\nabla_\theta(-2\theta^T X^T y) = -2X^T Y \qquad (P24)$$

$$\nabla_\theta(y^T y) = 0$$

$$\Rightarrow \nabla_\theta J(\theta) = X^T X\theta - X^T Y$$

$$
\begin{aligned}
\nabla_\theta J(\theta) &= X^T X \theta - X^T Y \\
&= X^T (X\theta - Y) \\
&= X^T \left(\underset{n \times p}{\begin{bmatrix} - x_1^T - \\ - x_2^T - \\ \cdots \\ - x_n^T - \end{bmatrix}} \underset{p \times 1}{\theta} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right) \\
&= \underset{p \times n}{X^T} \underset{n \times 1}{\begin{bmatrix} x_1^T \theta - y_1 \\ x_2^T \theta - y_2 \\ \cdots \\ x_n^T \theta - y_n \end{bmatrix}} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} x_1^T \theta - y_1 \\ \vdots \\ x_n^T \theta - y_n \end{bmatrix} \\
&= \sum_{i=1}^{n} x_i \left(x_i^T \theta - y_i \right)
\end{aligned}
$$

$$
\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix}
$$

$$
\begin{aligned}
\nabla_\theta J(\theta) &= X^T X \theta - X^T Y \\
&= X^T (X\theta - Y) \\
&= X^T \left(\underset{n \times p}{\begin{bmatrix} - x_1^T - \\ - x_2^T - \\ \cdots \\ - x_n^T - \end{bmatrix}} \underset{p \times 1}{\theta} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right) \\
&= \underset{p \times n}{X^T} \underset{n \times 1}{\begin{bmatrix} x_1^T \theta - y_1 \\ x_2^T \theta - y_2 \\ \cdots \\ x_n^T \theta - y_n \end{bmatrix}} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} x_1^T \theta - y_1 \\ \vdots \\ x_n^T \theta - y_n \end{bmatrix} \\
&= \sum_{i=1}^{n} x_i \underbrace{\boxed{\left(x_i^T \theta - y_i \right)}}_{1*1}
\end{aligned}
$$

LR with batch GD

- Steepest descent / GD
  - Note that:

$$\theta_j^{t+1} = \theta_j^t + \alpha \sum_{i=1}^{n} (y_i - \vec{\mathbf{x}}_i^T \theta^t) x_i^j$$

Update Rule Per Feature Variable-Wise

$$\nabla_\theta J = \left[\frac{\partial}{\partial \theta_1} J, \ldots, \frac{\partial}{\partial \theta_k} J \right]^T = -\sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \theta) \mathbf{x}_i$$

Based on CMU Handout's Definition of Gradient

$$\theta^{t+1} = \theta^t + \alpha \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \theta^t) \mathbf{x}_i$$

– This is as a **batch** gradient descent algorithm

Today

- ❑ A Practical Application of Regression Model
- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient Descent
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD (SGD)
  - ❑ Newton's method

LR with Stochastic GD ➔

- Batch GD rule:

$$\theta^{t+1} = \theta^t + \alpha \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \theta^t)\mathbf{x}_i$$

- Therefore, for a single training point (i), we have:

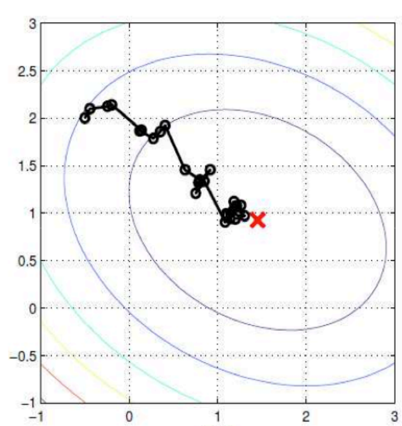
$$\theta^{t+1} = \theta^t + \alpha(y_i - \vec{\mathbf{x}}_i^T \theta^t)\vec{\mathbf{x}}_i$$

  – This is known as the Least-Mean-Square update rule, or the Widrow-Hoff learning rule
  – This is actually a "**stochastic**", "**coordinate**" descent algorithm
  – This can be used as a **on-line** algorithm

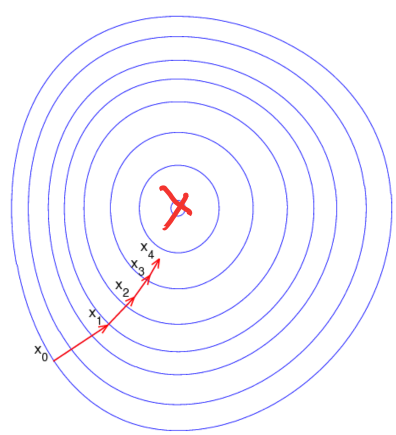
$$\theta_j^{t+1} = \theta_j^t + \alpha(y_i - \mathbf{x}_i^T \theta^t)x_{i,j}$$

Stochastic gradient descent / Online Learning Algorithm

SGD **GD**



versus



Stochastic gradient descent : More variations

- Single-sample:

$$\theta^{t+1} = \theta^t + \alpha (y_i - \vec{x}_i^T \theta^t) \vec{x}_i$$

- Mini-batch:

$$\theta^{t+1} = \theta^t + \alpha \sum_{j=1}^{B} (y_j - \vec{x}_j^T \theta^t) \vec{x}_j$$

e.g. $B = 15$

Stochastic gradient descent

SGD can also be used for offline learning, by repeatedly cycling through the data; each such pass over the whole dataset is called an **epoch**. This is useful if we have **massive datasets** that will not fit in main memory. In this offline case, it is often better to compute the gradient of a **mini-batch** of B data cases. If $B = 1$, this is standard SGD, and if $B = N$, this is standard steepest descent. Typically $B \sim 100$ is used.

Intuitively, one can get a fairly good estimate of the gradient by looking at just a few examples. Carefully evaluating precise gradients using large datasets is often a waste of time, since the algorithm will have to recompute the gradient again anyway at the next step. It is often a better use of computer time to have a noisy estimate and to move rapidly through parameter space.

SGD is often less prone to getting stuck in shallow local minima, because it adds a certain amount of "noise". Consequently it is quite popular in the machine learning community for fitting models such as neural networks and deep belief networks with non-convex objectives.

Nando de Freitas's tutorial slide

Summary so far: three ways to learn LR

- Normal equations $\theta^* = \left(X^T X\right)^{-1} X^T \vec{y}$
  - Pros: a single-shot algorithm! Easiest to implement.
  - Cons: need to compute pseudo-inverse $(X^TX)^{-1}$, expensive, numerical issues (e.g., matrix is singular ..), although there are ways to get around this …
- GD or Steepest descent $\theta^{t+1} = \theta^t + \alpha \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \theta^t) \mathbf{x}_i$
  - Pros: easy to implement, conceptually clean, guaranteed convergence
  - Cons: batch, often slow converging
- Stochastic LMS update rule $\theta^{t+1} = \theta^t + \alpha (y_i - \vec{\mathbf{x}}_i^T \theta^t) \vec{\mathbf{x}}_i$
  - Pros: on-line, low per-step cost, fast convergence and perhaps less prone to local optimum
  - Cons: convergence to optimum not always guaranteed

Direct (normal equation) vs. Iterative (GD) methods

- Direct methods: we can achieve the solution in a single step by solving the normal equation
  - Using Gaussian elimination or QR decomposition, we converge in a finite number of steps
  - It can be infeasible when data are streaming in in real time, or of very large amount
- Iterative methods: stochastic or steepest gradient
  - Converging in a limiting sense
  - But more attractive in large practical problems
  - Caution is needed for deciding the learning rate a

Evaluation : for Regression Models

$$J(\theta)=\frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i(\vec{x}_i)-y_i)^2$$

Sum of squared error (SSE) on training set

- Testing MSE (mean-squared-error) to report:

$$MSE_{test}=\frac{1}{m}\sum_{i=n+1}^{n+m}(\mathbf{x}_i^T\theta^*-y_i)^2$$

- Training MSE to report:

$$MSE_{train}=\frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta^*-y_i)^2$$

Convergence rate

- **Theorem**: the steepest descent equation algorithm converge to the minimum of the cost characterized by normal equation:

$$\theta^{(\infty)}=(X^TX)^{-1}X^Ty$$

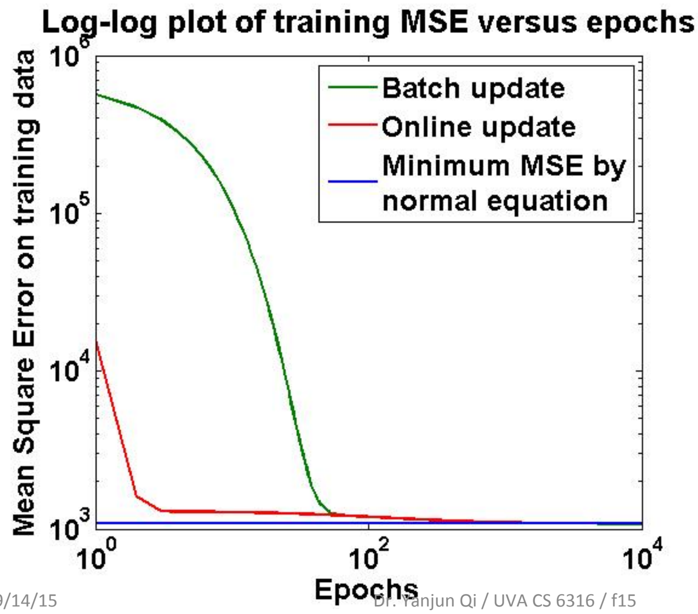
If the learning rate parameter satisfy ➔

$$0<\alpha<2/\lambda_{\max}[X^TX]$$

- A formal analysis of GD-LR need more math; in practice, one can use a small a, or gradually decrease a.

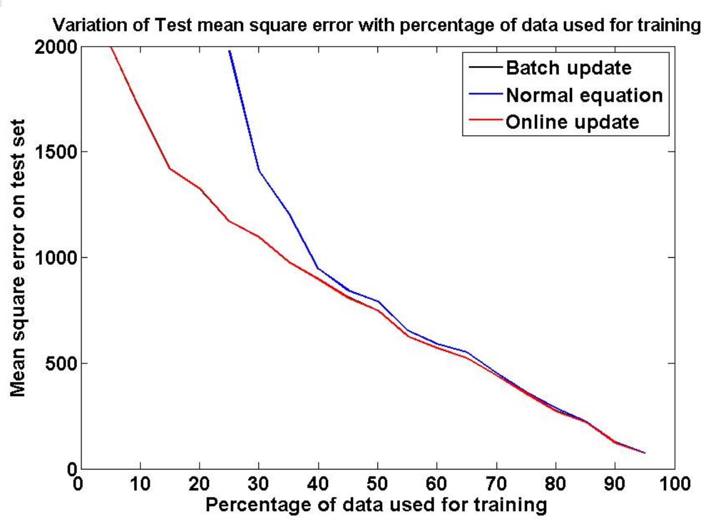
$\alpha_0 = 0.05$

Convergence Curves, for an example



- For the batch method, the training MSE is initially large due to uninformed initialization
- In the online update, N updates for every epoch reduces MSE to a much smaller value.

Performance vs. Training Size for an example



- The results from B and O update are almost identical. So the plots coincide.
- The test MSE from the normal equation is more than that of B and O during small training. This is probably due to overfitting.
- In B and O, since only 2000 (for example) iterations are allowed at most. This roughly acts as a mechanism that avoids overfitting.

Today

- ❑ A Practical Application of Regression Model
- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient Descent
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD (SGD)
  - ❑ Newton's method

Dr. Yanjun Qi / UVA CS 6316 / f15

Review: Convex function

- Intuitively, a convex function (1D case) has a single point at which the derivative goes to zero, and this point is a minimum.
- Intuitively, a function f (1D case) is convex on the range [a,b] if a function's second derivative is positive every-where in that range.
- Intuitively, if a function's Hessians is psd (positive semi-definite!), this (multivariate) function is Convex
  - Intuitively, we can think "Positive definite" matrices as analogy to positive numbers in matrix case

Newton's method for optimization

- The most basic second-order optimization algorithm
- Updating parameter with

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_K^{-1}\mathbf{g}_k$$

Review: Hessian Matrix / n==2 case

Singlevariate → multivariate

$$f(x,y)$$

- 1st derivative to gradient, $g = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$
- 2nd derivative to Hessian $H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$

Review: Hessian Matrix

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a function that takes a vector in $\mathbb{R}^n$ and returns a real number. Then the ***Hessian*** matrix with respect to x, written $\nabla_x^2 f(x)$ or simply as H is the $n \times n$ matrix of partial derivatives,

$$\nabla_x^2 f(x) \in \mathbb{R}^{n \times n} = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}.$$

Newton's method for optimization

- Making a quadratic/second-order Taylor series approximation

$$\hat{f}_{quad}(\boldsymbol{\theta}) = f(\boldsymbol{\theta}_k) + \mathbf{g}_k^T(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T \mathbf{H}_k (\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

Finding the minimum solution of the above right quadratic approximation (quadratic function minimization is easy !)

$$\hat{f}(\theta) = f(\theta_k) + g_k^T(\theta - \theta_k) + \underbrace{\frac{1}{2}(\theta - \theta_k)^T H_k (\theta - \theta_k)}$$

$$\Downarrow$$

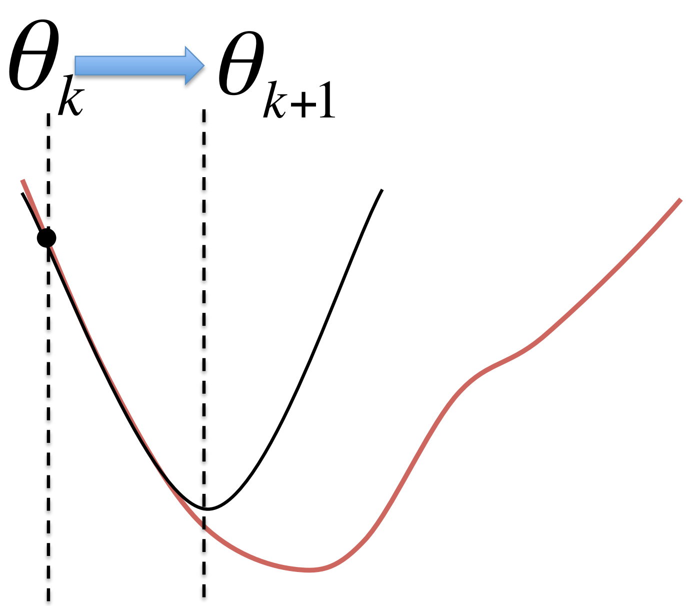
$$\frac{1}{2}\left(\theta^T H_k \theta - 2\theta^T H_k \theta_k + \theta_k^T H_k \theta_k\right)$$

$$\frac{\partial \hat{f}(\theta)}{\partial \theta} = 0 + g_k + \underbrace{\frac{2}{2} H_k \theta - \frac{2}{2} H_k \theta_k}_{\text{see p24 handout}} := 0$$

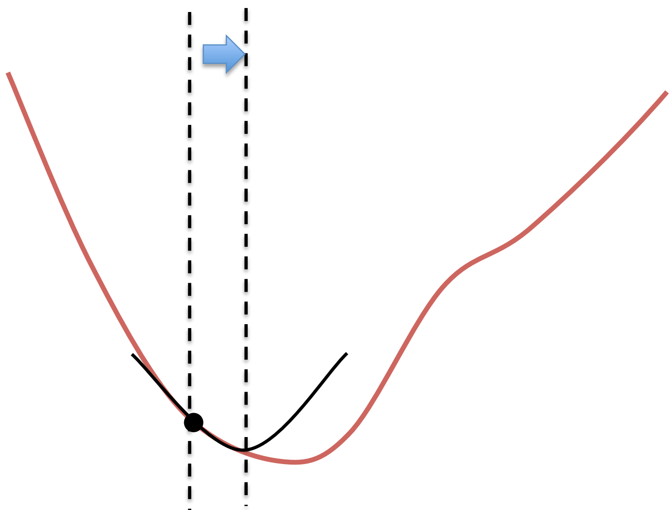
$$g_k + H_k(\theta - \theta_k) = 0$$

$$\Rightarrow \theta = \theta_k - H_k^{-1} g_k \quad \text{where } \begin{cases} H_k \in R^{p \times p} \\ g_k \in R^p \end{cases}$$

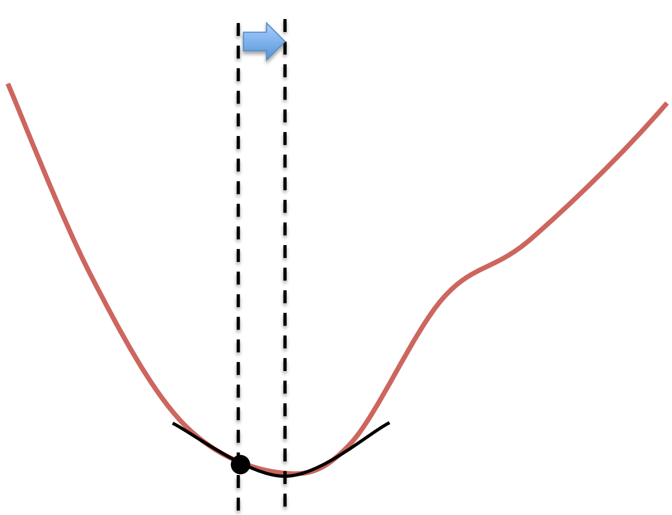
Newton's Method / second-order Taylor series approximation



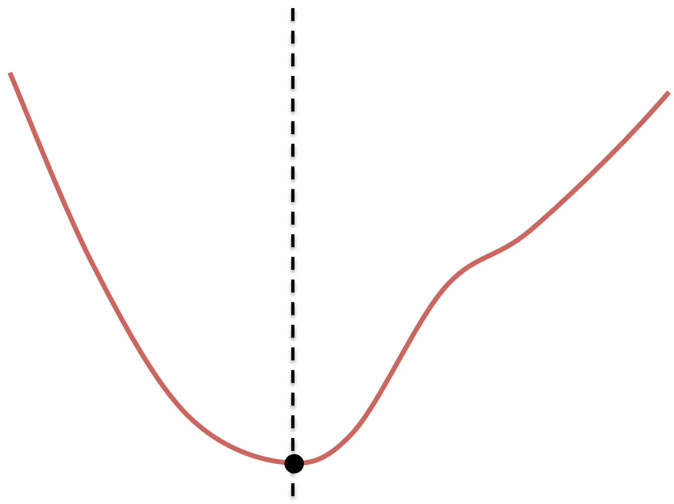
Newton's Method / second-order Taylor series approximation



Newton's Method / second-order Taylor series approximation



Newton's Method / second-order Taylor series approximation



Newton's Method

- At each step:

$$\theta_{k+1} = \theta_k - \frac{f'(\theta_k)}{f''(\theta_k)}$$

$$\theta_{k+1} = \theta_k - H^{-1}(\theta_k)\nabla f(\theta_k)$$

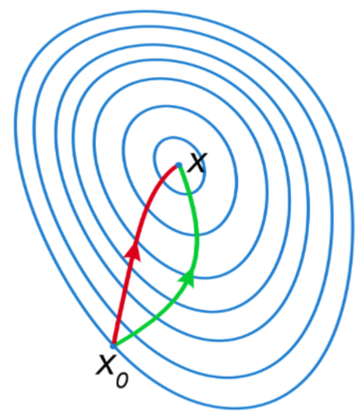
- Requires 1st and 2nd derivatives
- Quadratic convergence
- ➔ However, finding the inverse of the Hessian matrix is often expensive

Comparison

- Newton's method vs. Gradient descent

A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes).

Newton's method uses curvature information to get a more direct route ...



$$J(\theta) = \frac{1}{2}(y - X\theta)^T(y - X\theta)$$

$$\nabla_\theta J(\theta) = X^T X \theta - X^T \vec{y}$$

$$H = \nabla_\theta^2 J(\theta) = X^T X$$

$$\Rightarrow \theta^t = \theta^{t-1} - H^{-1} \nabla J(\theta^{t-1})$$

$$= \theta^{t-1} - \underbrace{(X^T X)^{-1}} \left[\underbrace{X^T X} \theta^{t-1} - X^T \vec{y} \right]$$

$$= \left[\theta^{t-1} - \theta^{t-1}\right] + (X^T X)^{-1} X^T \vec{y}$$

$$= (X^T X)^{-1} X^T \vec{y}$$

WHY ??? Normal Eq?

Newton's method for Linear Regression

Today Recap

- ❑ A Practical Application of Regression Model
- ❑ More ways to train / perform optimization for linear regression models
  - ❑ Gradient Descent
  - ❑ Gradient Descent (GD) for LR
  - ❑ Stochastic GD (SGD)
  - ❑ Newton's method

References

- Big thanks to Prof. Eric Xing @ CMU for allowing me to reuse some of his slides
- ❑ **Notes about Gradient Descent from Toussaint: (please read) http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gradientDescent.pdf**
- ❑ http://en.wikipedia.org/wiki/Matrix_calculus
- ❑ Prof. Nando de Freitas's tutorial slide