UVA CS 4501: Machine Learning

Lecture 4: More optimization for Linear Regression

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Where are we? ➔
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
  - ~~Connecting to Newton's method~~

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

A little bit more about [Optimization]

- Objective function $F(x)$ → $J(\theta)$
- Variables x → θ
- Constraints → $\theta \in R^p$

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

Method I: normal equations

- Write the cost function in matrix form:

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

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

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

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

The normal equations

$$\Downarrow$$

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

X^TX Gram matrix

↓

PSD

↓

$J(\theta)$ convex

↓

$\nabla J(\theta) = 0 \Rightarrow \theta^*$

Today

- More ways to train / perform optimization for linear regression models
  - Review: Gradient Descent
  - Gradient Descent (GD) for LR
  - Stochastic GD (SGD) for LR

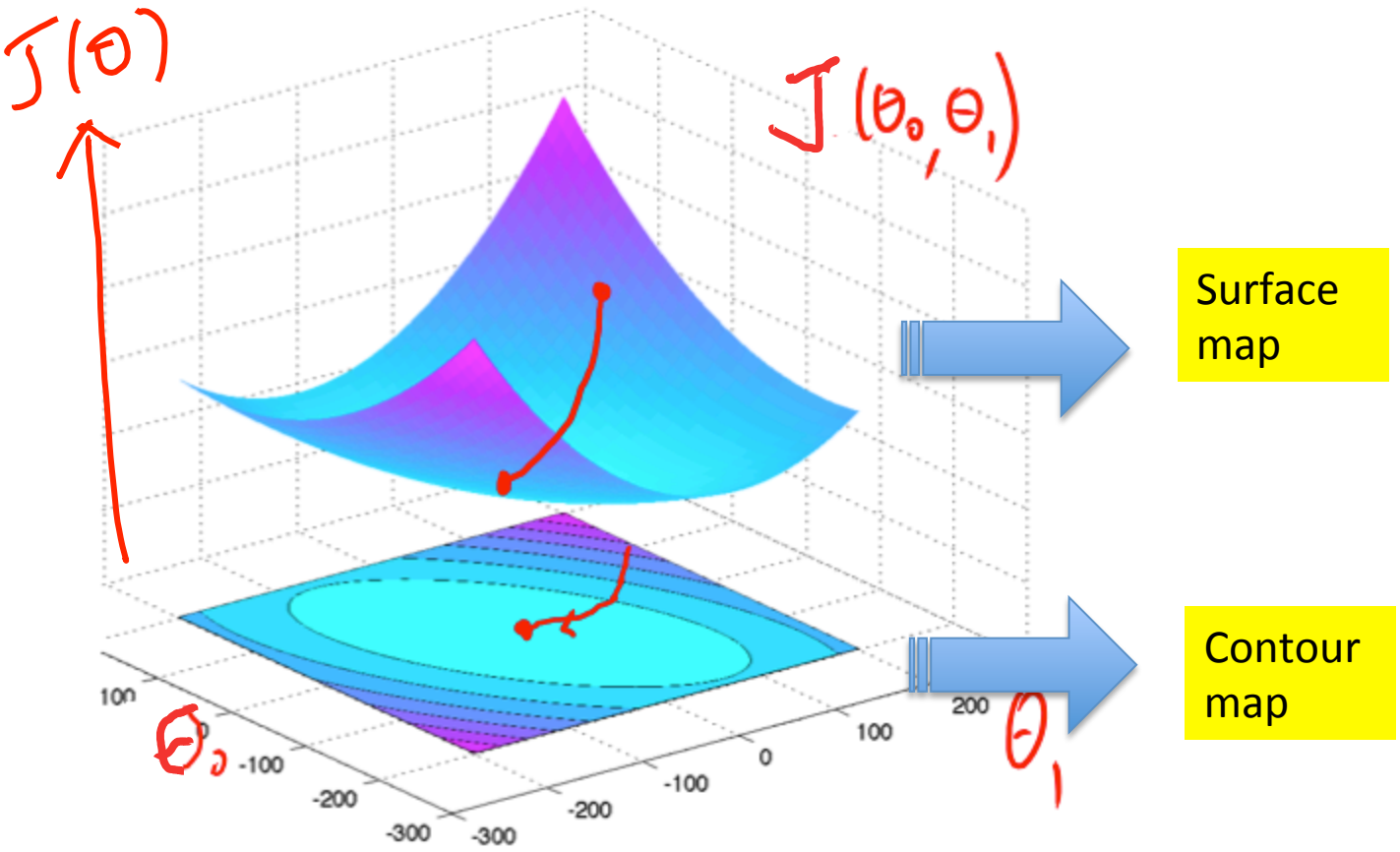
Review: Definitions of gradient (from L2-note page 21)

- Size of gradient is always the same as the size of the variable/parameter

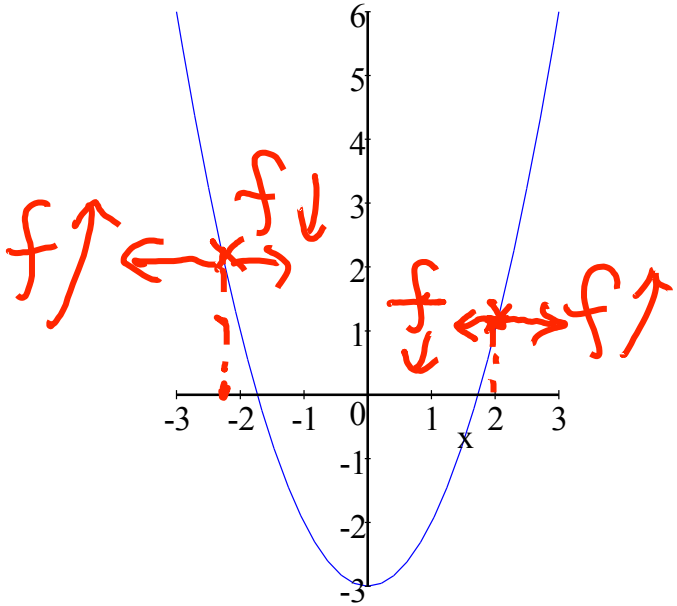
$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \quad \text{if } x \in \mathbb{R}^n$$

A vector whose entries respectively contain the n partial derivatives

Two ways of Illustrating the Objective Function (2D case)



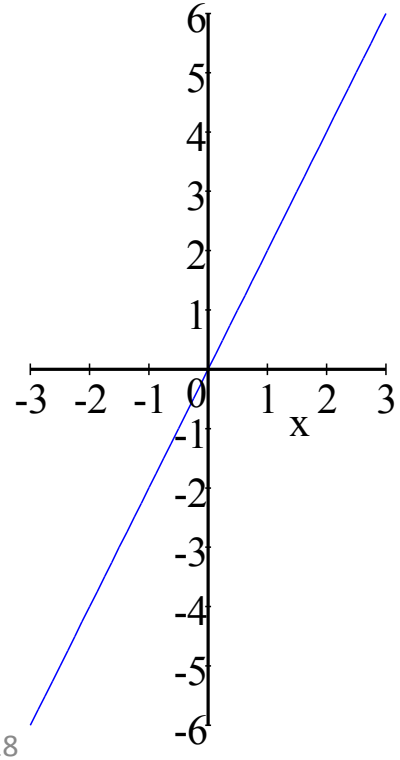
Review: Derivative of a Quadratic Function



$$y = x^2 - 3$$

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

$$y' = 2x$$



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)

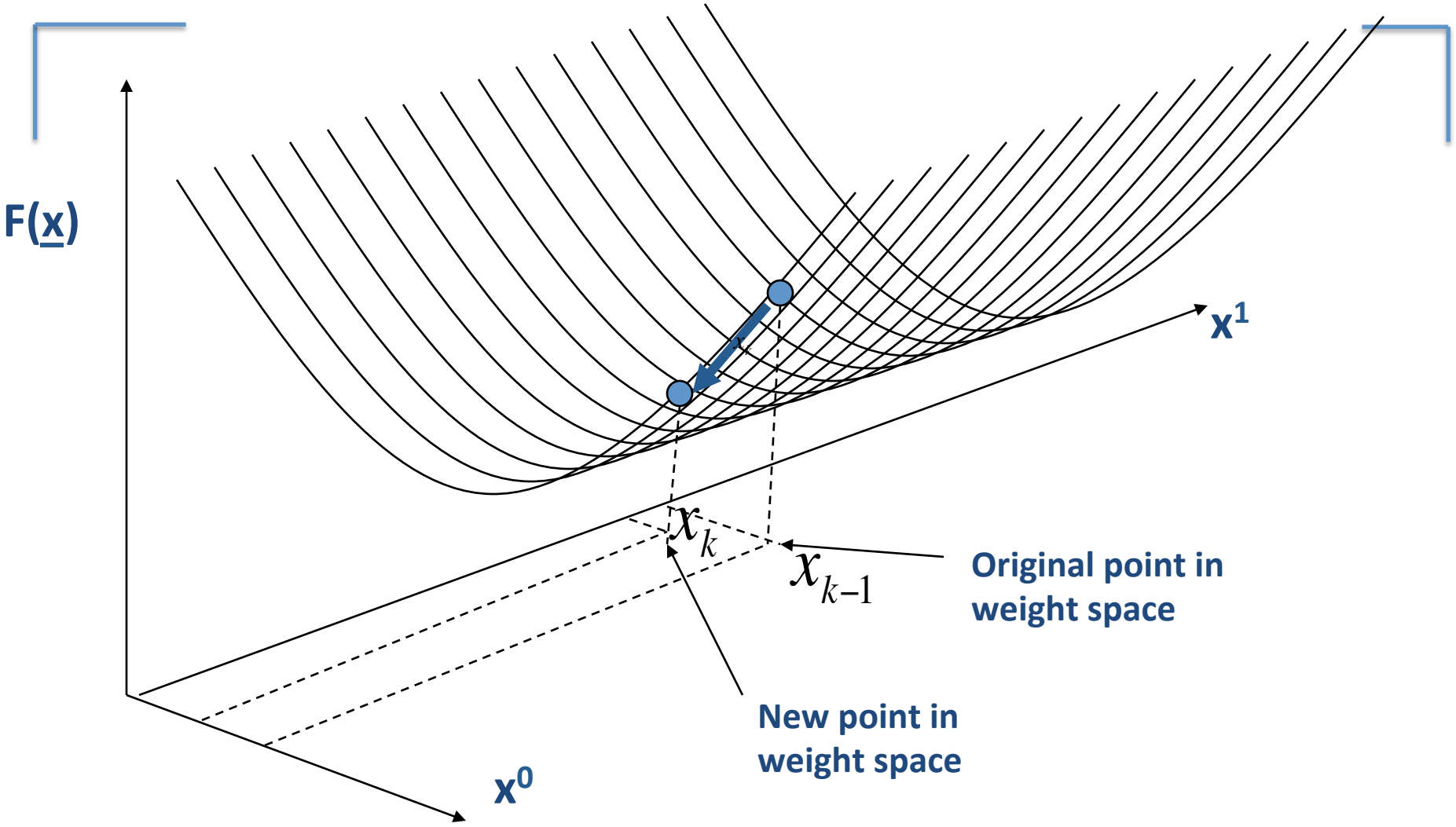
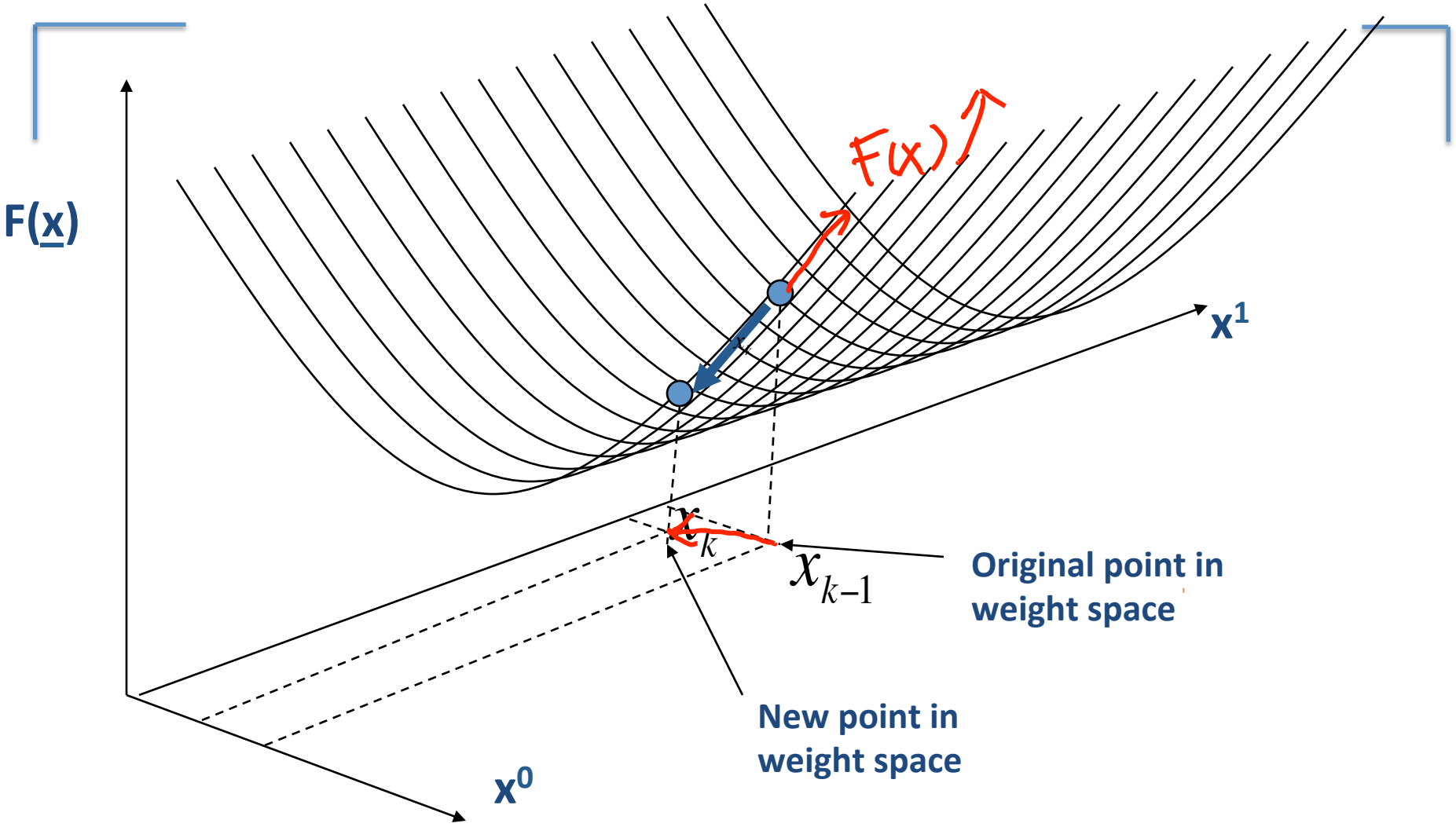
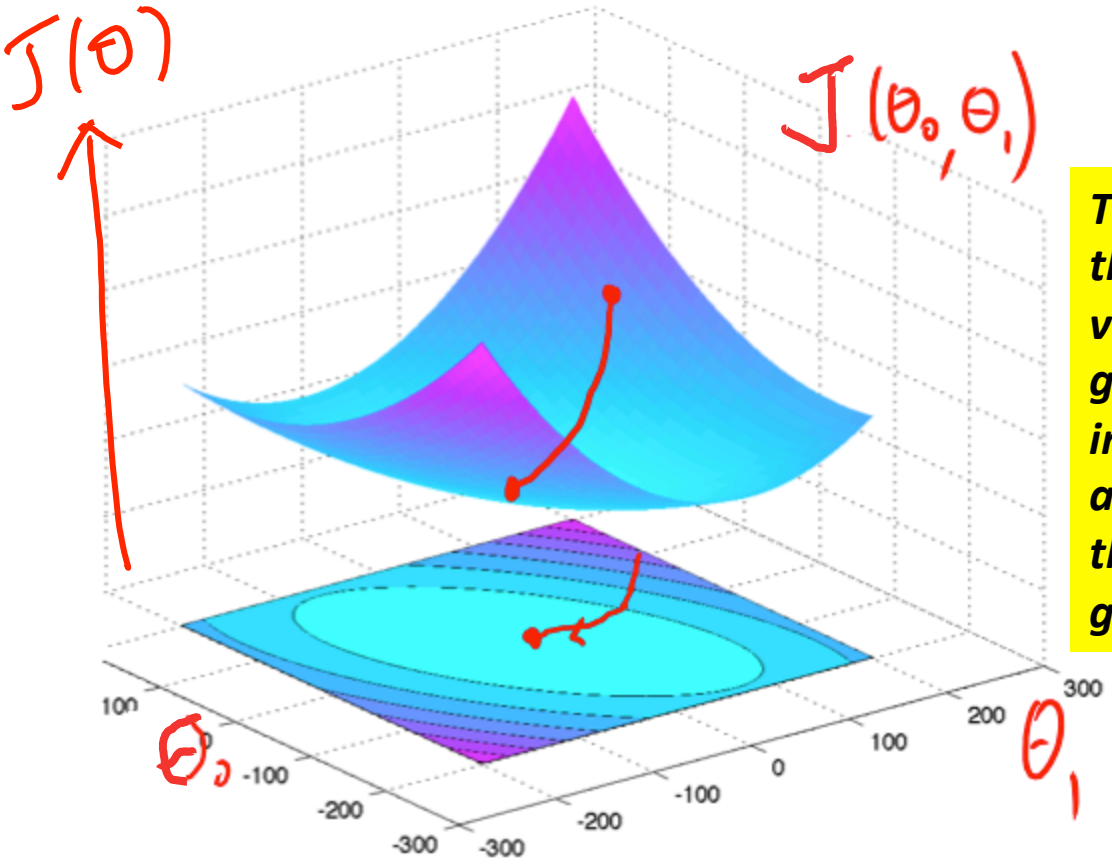


Illustration of Gradient Descent (2D case)



Two ways of Illustrating the Objective Function and Gradient Descent (variable is 2D case)



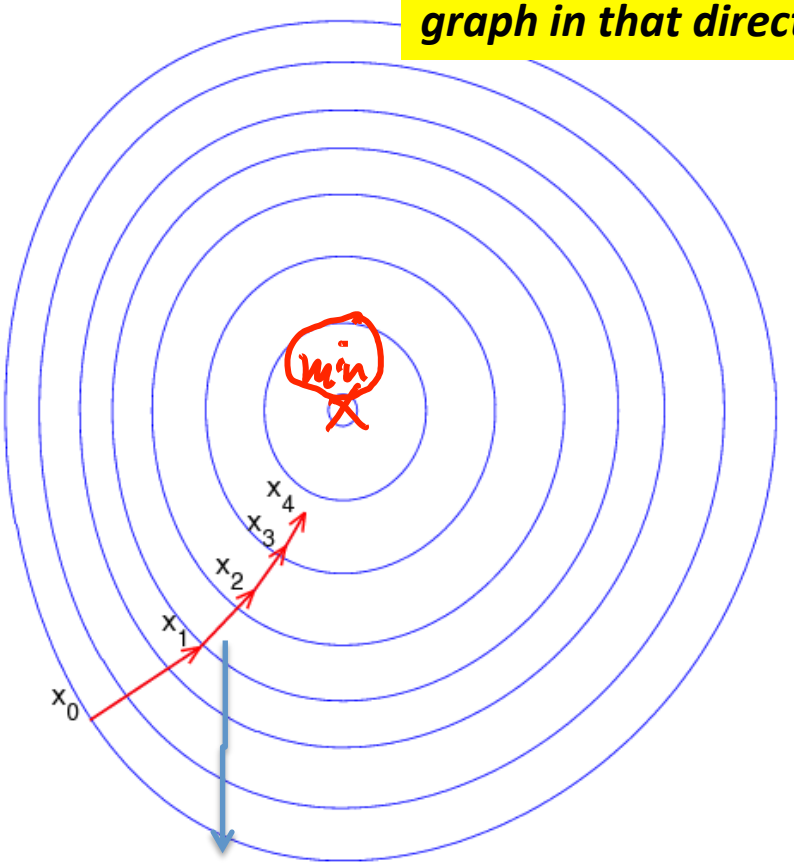
The gradient points in the direction (in the variable space) of the greatest rate of increase of the function and its magnitude is the slope of the surface graph in that direction

e.g. Gradient Descent (Steepest Descent) – contour map view

The gradient (in the variable space) points in the direction of the greatest rate of increase of the function and its magnitude is the slope of the surface graph in that direction

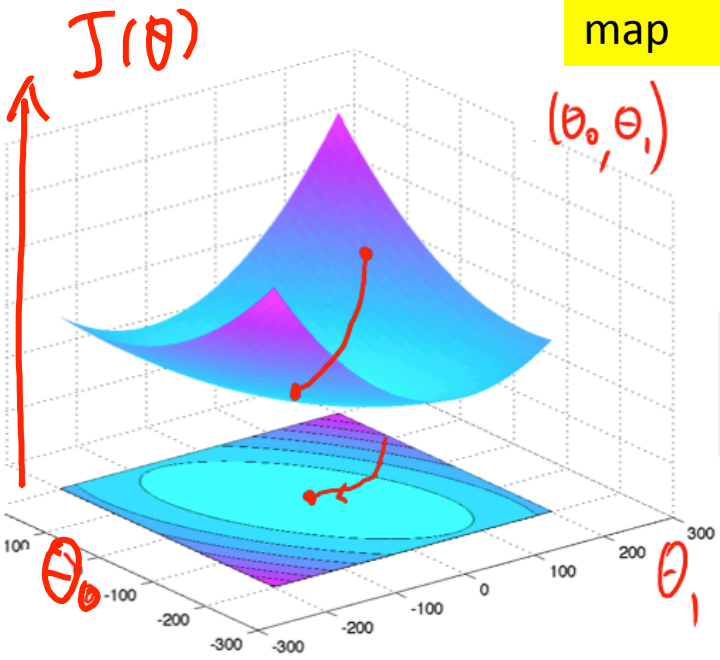
A first-order optimization algorithm.

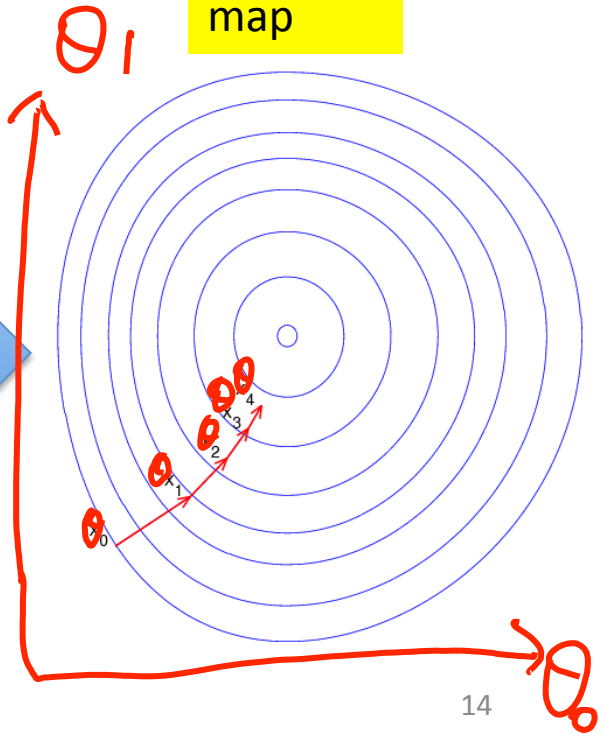
To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient of the function at the current point.



Surface map

Contour map



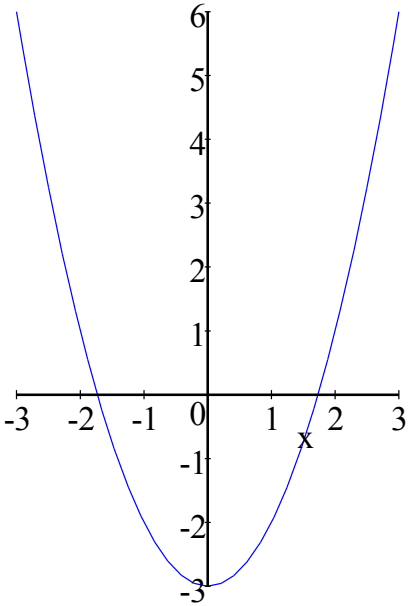


Gradient Descent (GD): An iterative Algorithm

- Initialize k=0, (randomly or by prior) choose x_0
- While k<k_{max} For the k-th epoch

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

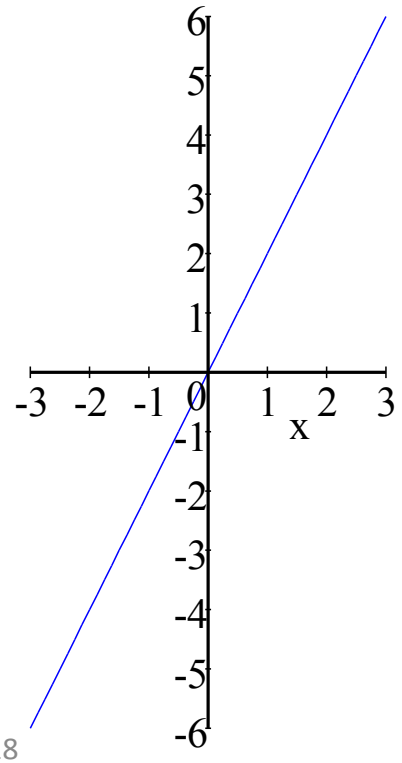
Review: Derivative of a Quadratic Function



$$F(x) = x^2 - 3$$

$$\nabla_x F(x) = F'(x) = 2x$$

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



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

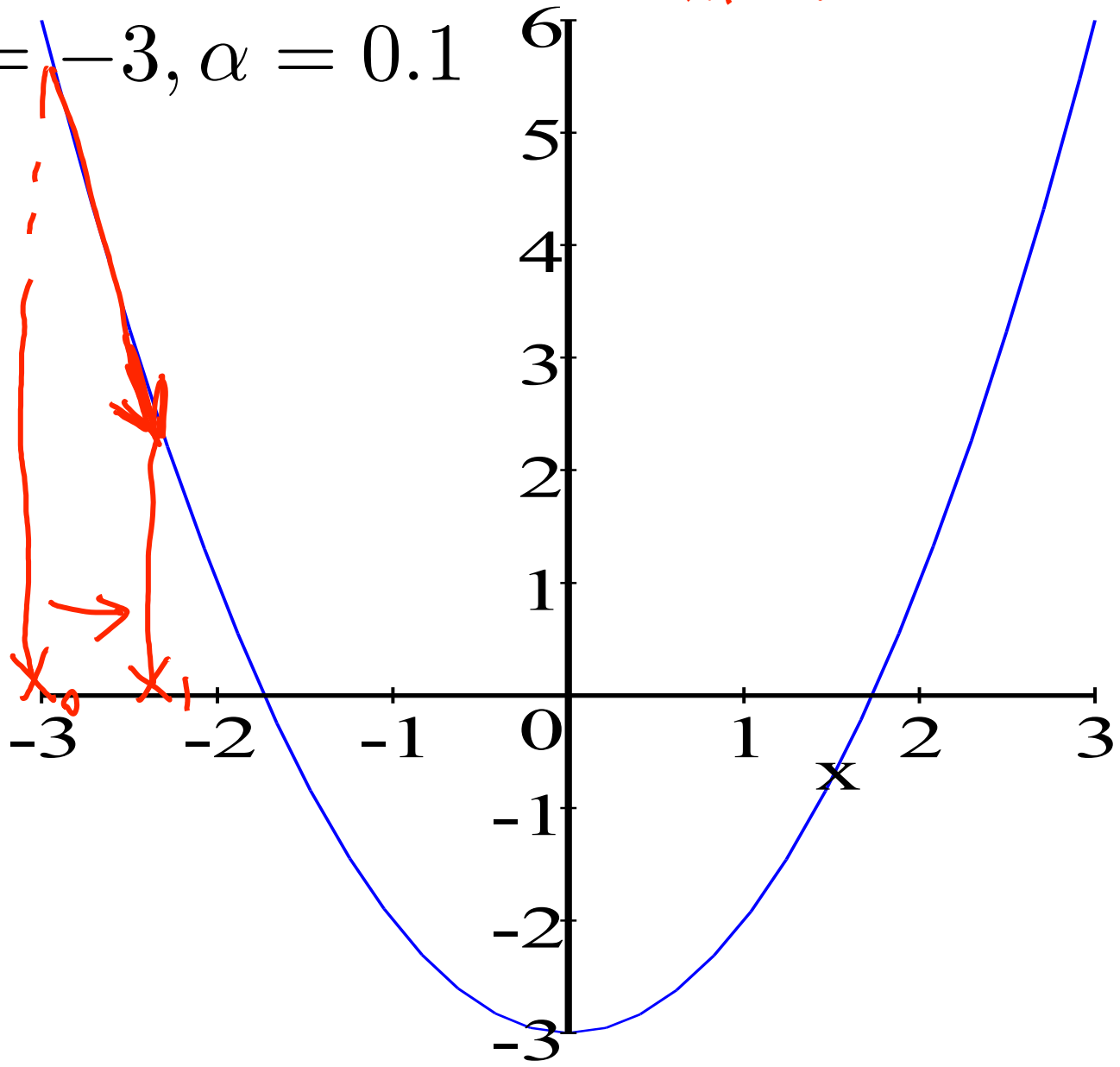
$$x_0 = -3, \alpha = 0.1$$

$x_1 = x_0 - \alpha \nabla F(x_0)$

$= -3$

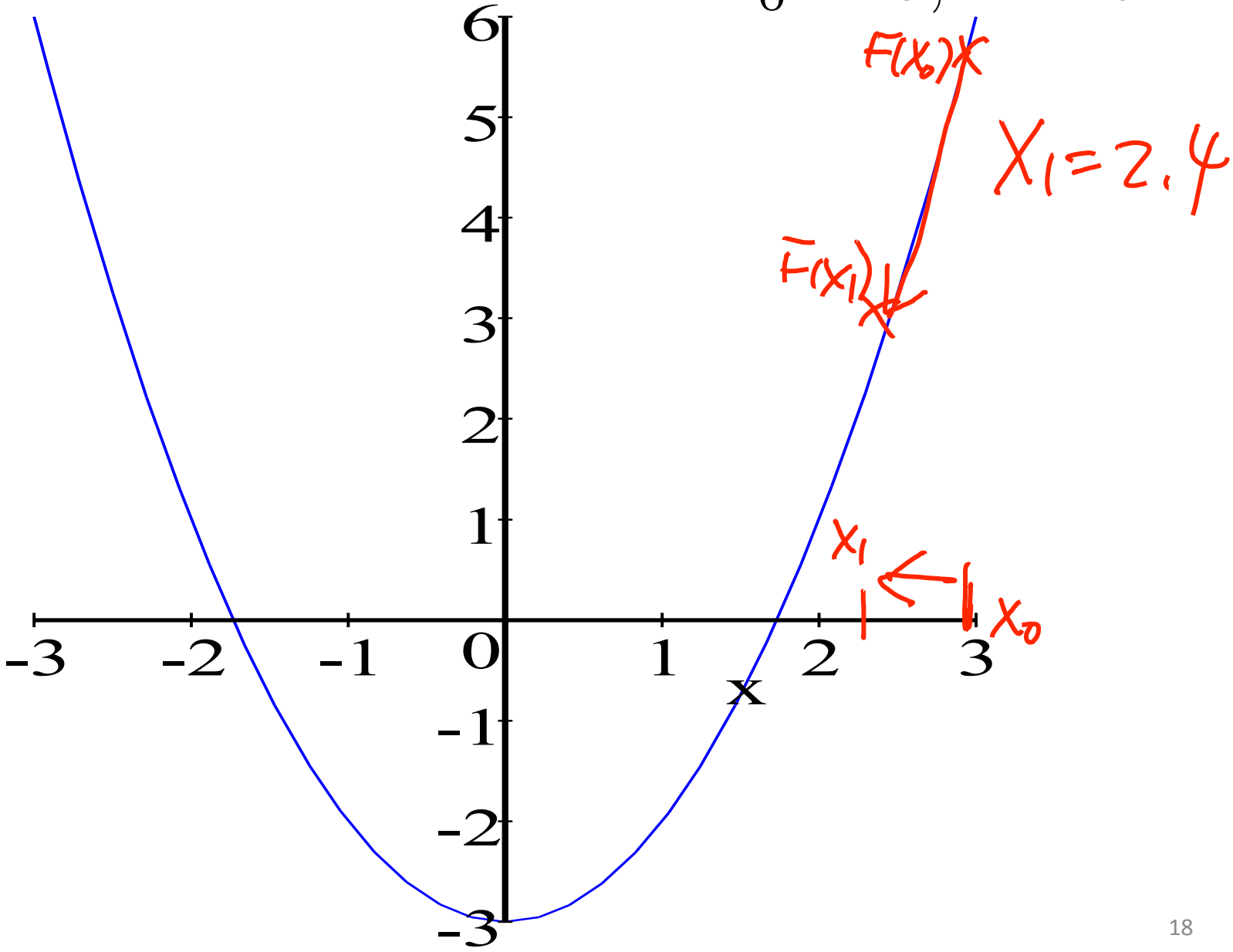
$-2 \times 0.1 \times (-3)$

$= -2.4$



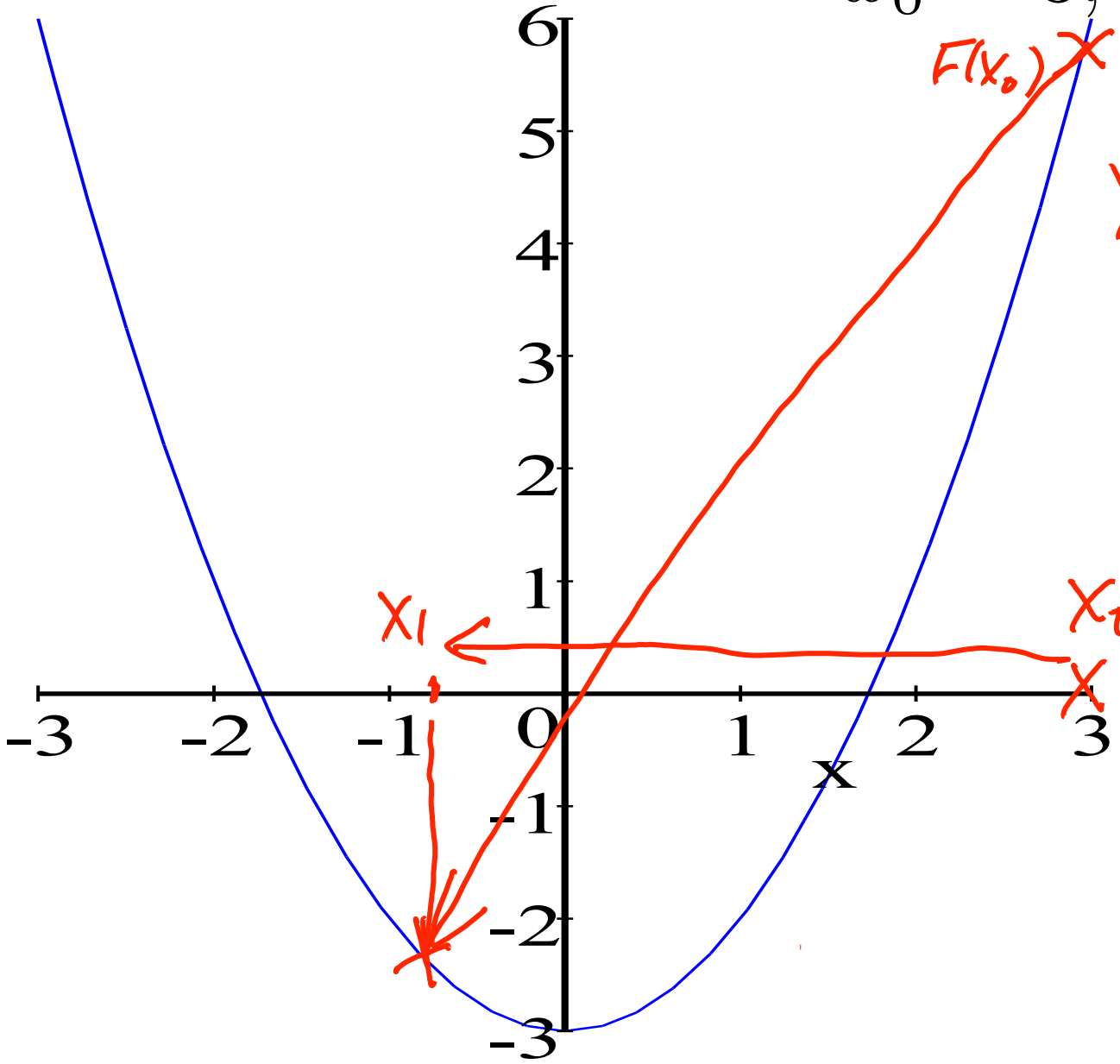
$$x_k = x_{k-1} - \alpha \nabla_x F(x_{k-1})$$

$$x_0 = 3, \alpha = 0.1$$



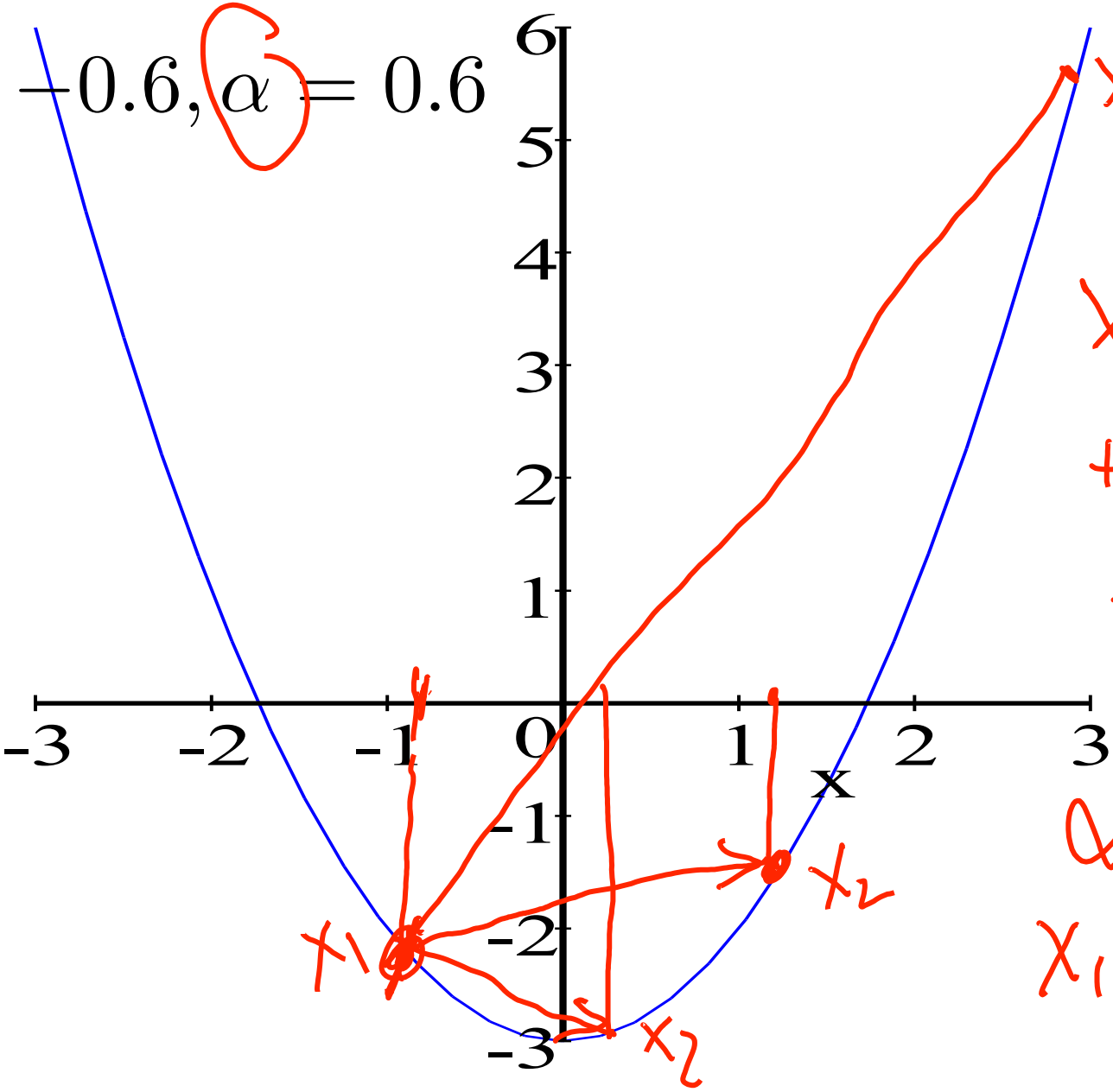
$$x_k = x_{k-1} - \alpha \nabla_x F(x_{k-1})$$

$$x_0 = 3, \alpha = 0.6$$



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

$$x_0 = -0.6, \alpha = 0.6$$



$x_1 = -0.6$

$+2 \times 0.6 \times 0.6$

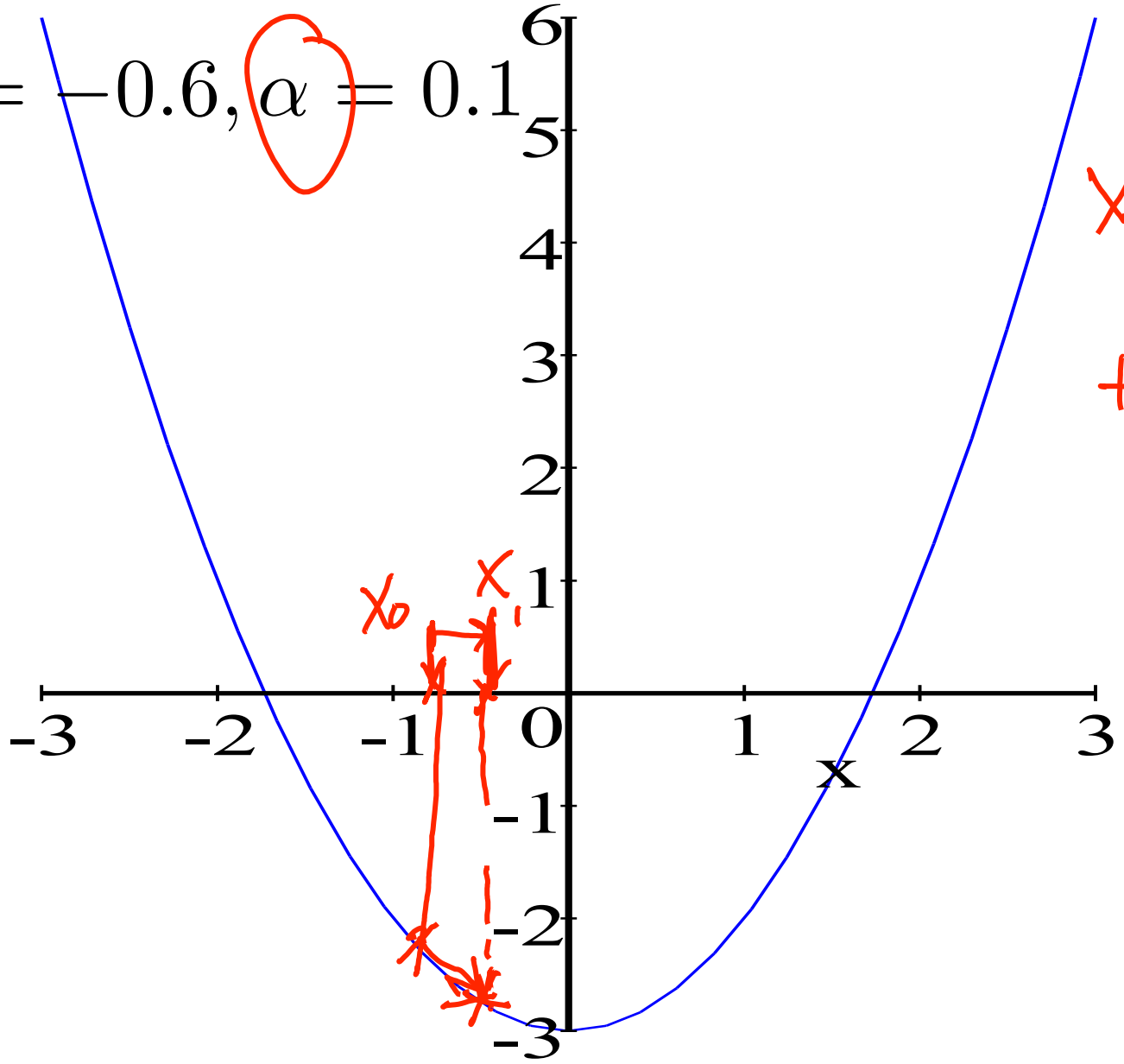
$= 0.12$

$\alpha = 1.5$

$x_1 = 1.2$

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

$$x_0 = -0.6, \alpha = 0.1$$



$X_1 = -0.6$

$+2 \times 0.1 + 0.6$

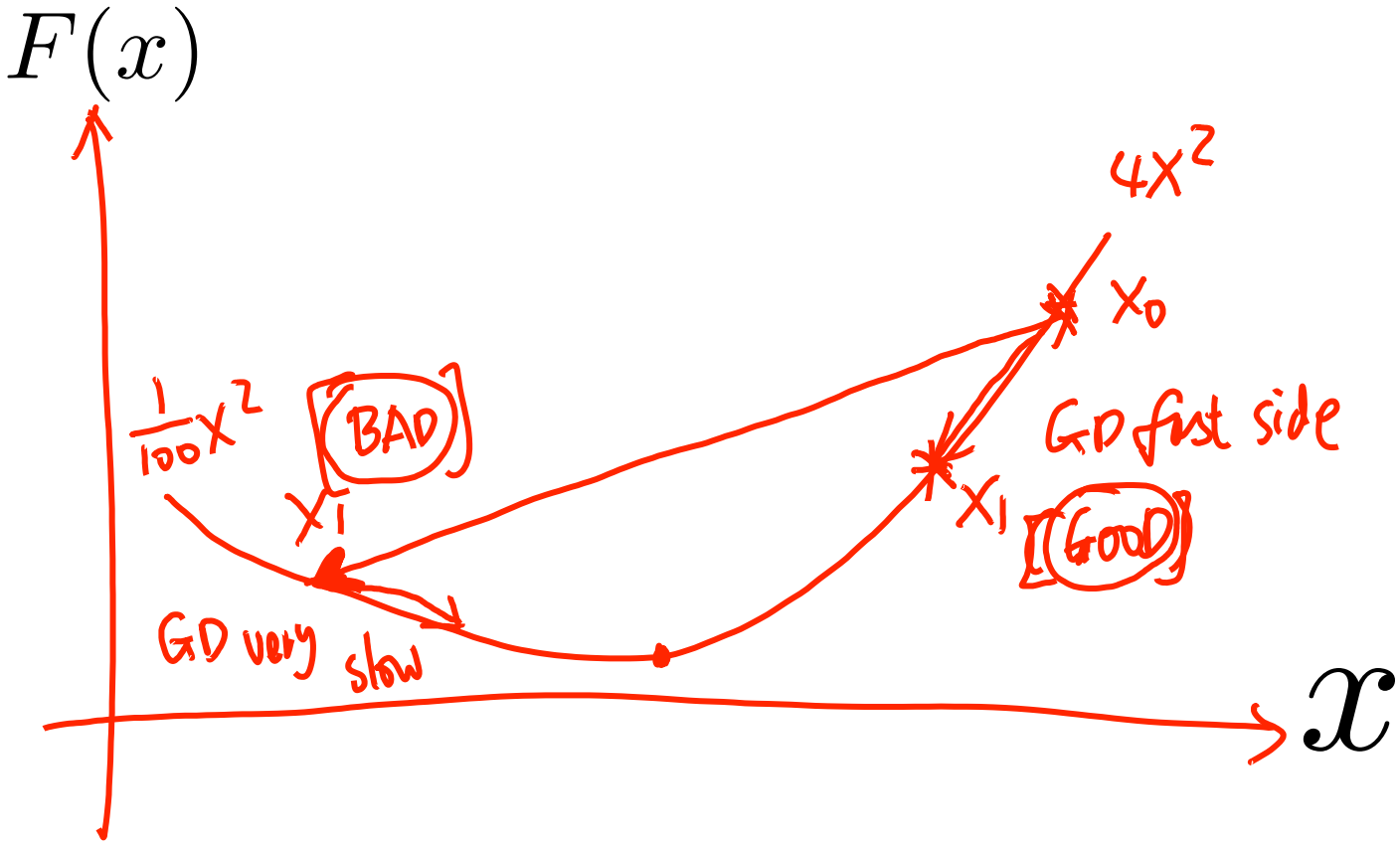
$= -0.48$

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

$$x_0 = 3, \alpha = 0.1$$

$x_1 = 3 - 4 \times 0.1 \times 3$

$= 1.8$


$F(x)$
$4x^2$
x_0
$\frac{1}{100}x^2$
BAD
x_1
GD fast side
x_1
GOOD
GD very slow
x

X_t	α	X_(t+1)	$f(x)$
-3	0.1	-2.4	x^2
3	0.1	2.4	x^2
3 -0.6	0.6 0.6	-0.6 0.12	x^2
-0.6	0.1	-0.48	x^2
3	0.1	1.8	$2x^2$

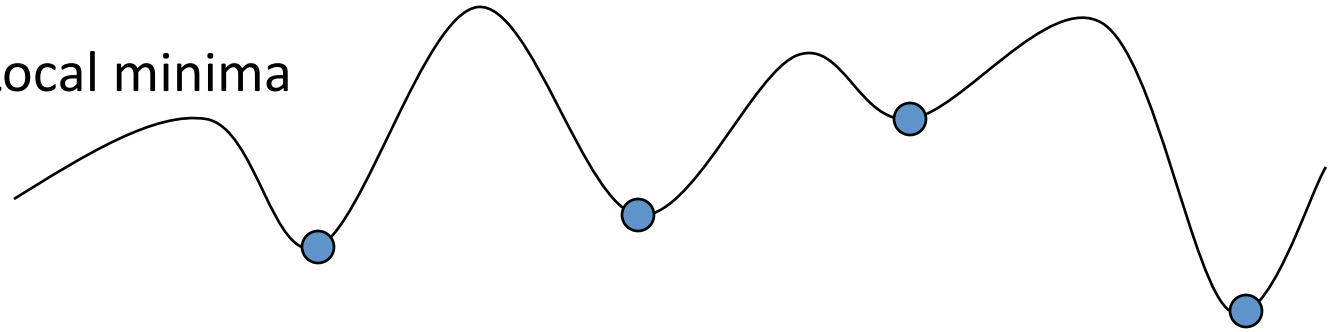
α

① Small

② smaller as t ↑

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

- More ways to train / perform optimization for linear regression models
  - Review: Gradient Descent
  - Gradient Descent (GD) for LR
  - Stochastic GD (SGD) for LR

Review: Loss function of LR

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

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

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

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

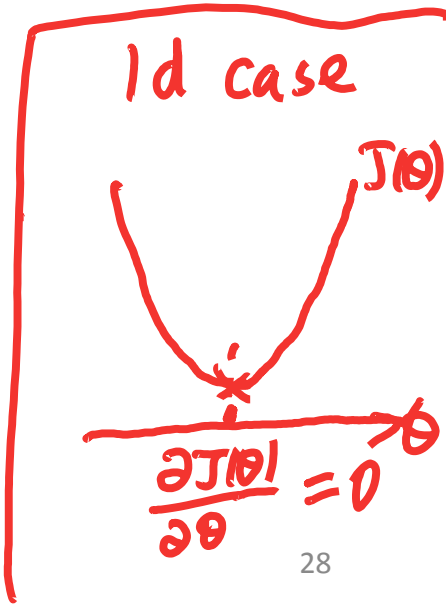
$$= \left(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y\right)\frac{1}{2}$$

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

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

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

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



See handout 4.1 + 4.3 ⇒ matrix calculus, partial deri ⇒ Gradient

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

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

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

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

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

$$= X^T (X\theta - Y)$$

$$= X^T \left(\begin{bmatrix} -\vec{x}_1^T- \\ -\vec{x}_2^T- \\ \vdots \\ -\vec{x}_n^T- \end{bmatrix}_{n\times p} \theta - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{p\times 1} \right)$$

$$= \underset{p\times n}{X^T} \begin{bmatrix} \vec{x}_1^T\theta - y_1 \\ \vec{x}_2^T\theta - y_2 \\ \cdots\cdots \\ \vec{x}_n^T\theta - y_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} | & | & & | \\ \vec{x}_1 & \vec{x}_2 & \cdots & \vec{x}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \vec{x}_1^T\theta - y_1 \\ \vdots \\ \vec{x}_n^T\theta - y_n \end{bmatrix}$$

$$= \sum_{i=1}^{n} \vec{x}_i \underbrace{(\vec{x}_i^T\theta - y_i)}_{\text{scalar}} = \sum (y_i - \vec{x}_i^T\theta)\vec{x}_i$$

LR with batch GD

- The Cost Function:

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

$$\vec{\theta}_{k+1} = \vec{\theta}_k - \alpha \nabla_\theta J(\vec{\theta}_k)$$

- Consider a **gradient descent** algorithm:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{p-1} \end{bmatrix}$$

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

$$\theta_j^{t+1} = \theta_j^t - \alpha\frac{\partial}{\partial\theta_j}J(\theta)\Big|_t$$

For the (t+1)-th epoch
For the j-th variable

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

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

$$= X^T (X\theta - Y)$$

$$= X^T \left(\begin{bmatrix} -x_1^T- \\ -x_2^T- \\ \vdots \\ -x_n^T- \end{bmatrix}_{n\times p} \theta - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{p\times 1} \right)$$

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

$$= \underset{p\times n}{X^T} \begin{bmatrix} x_1^T\theta - y_1 \\ x_2^T\theta - y_2 \\ \cdots \\ x_n^T\theta - y_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} 1 & 1 & & 1 \\ x_1 & x_2 & \cdots & x_n \\ 1 & 1 & & 1 \end{bmatrix} \begin{bmatrix} x_1^T\theta - y_1 \\ \vdots \\ x_n^T\theta - y_n \end{bmatrix}$$

$$= \sum_{i=1}^{n} x_i \underbrace{(x_i^T\theta - y_i)}_{1*1}$$

LR with batch GD

- Steepest descent as a **batch** GD algorithm
  - Note that: $\nabla_\theta J = \left[\frac{\partial}{\partial \theta_1} J, \ldots, \frac{\partial}{\partial \theta_k} J\right]^T = -\sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \theta)\mathbf{x}_i$

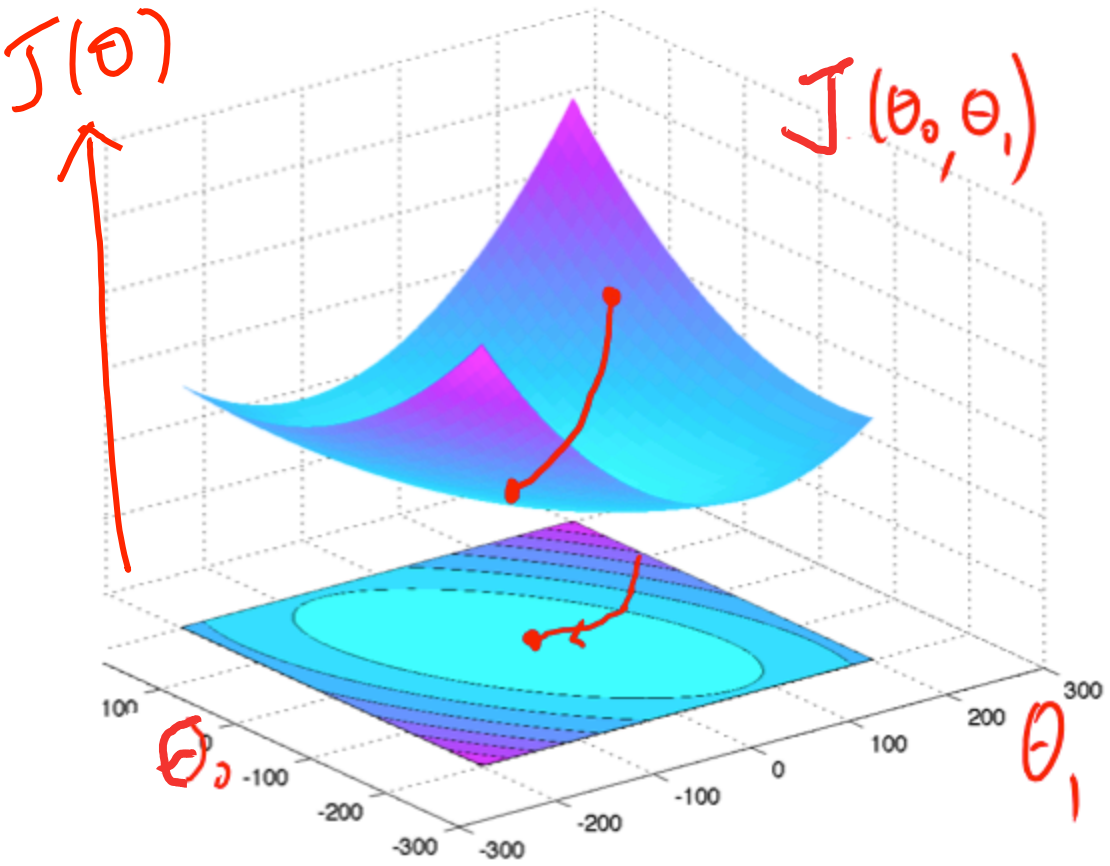
$$\theta^{t+1} = \theta^t + \alpha \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \theta^t)\mathbf{x}_i$$

  - For its j-th variable:

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

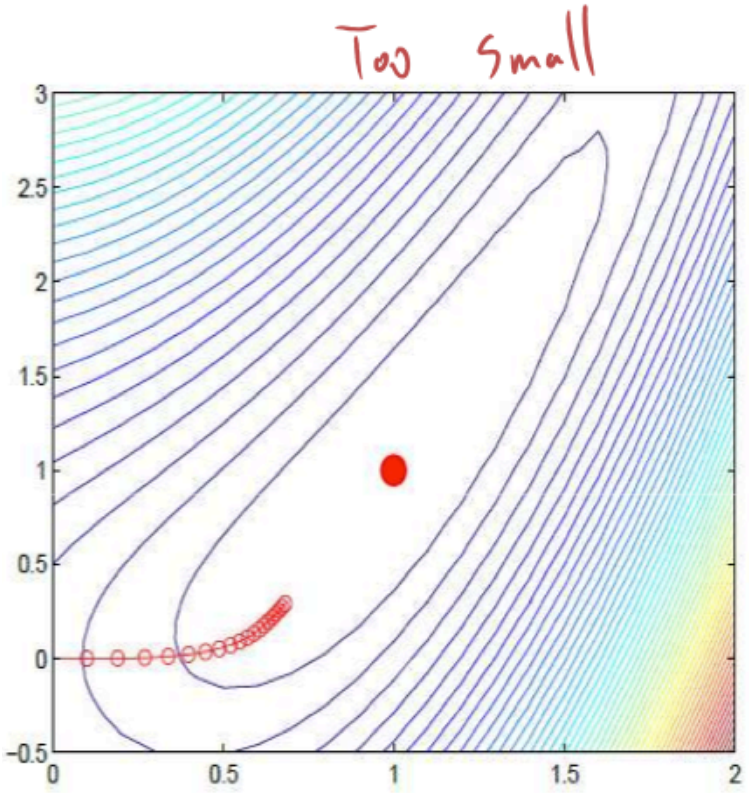
Update Rule Per Feature (Variable-Wise)

Illustration of Gradient Descent (2D case)

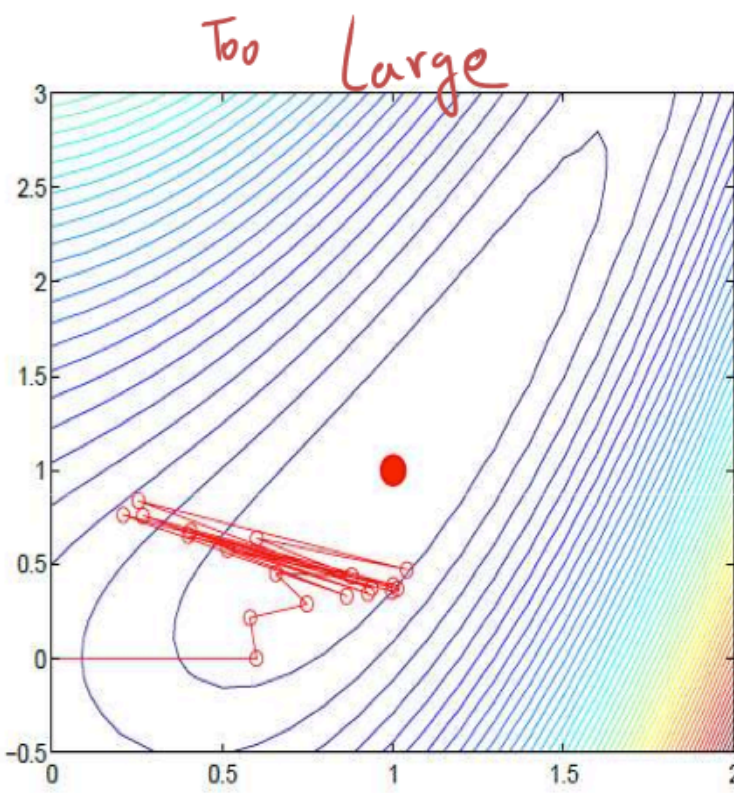


$y = \theta_0 + \theta_1 x_1$

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



$\alpha = 0.1.$



$\alpha = 0.6.$

Today

- More ways to train / perform optimization for linear regression models
  - Review: Gradient Descent
  - Gradient Descent (GD) for LR
  - Stochastic GD (SGD) for LR

LR with Stochastic GD ➔

- Batch GD rule:

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

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

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

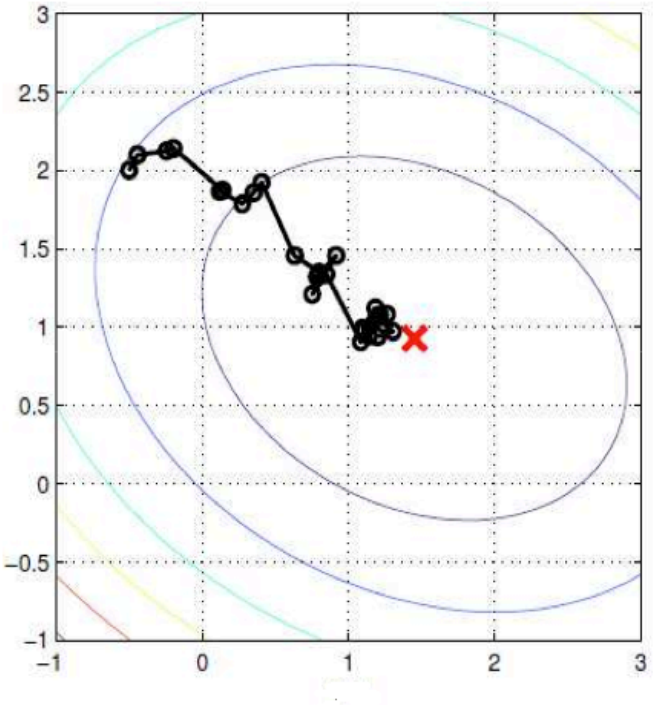
$$\theta^{t+1} = \theta^t + \alpha (y_i - \vec{\mathbf{x}}_i^T \theta^t) \vec{\mathbf{x}}_i$$

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

  - A "**stochastic**" descent algorithm, can be used as an **on-line** algorithm

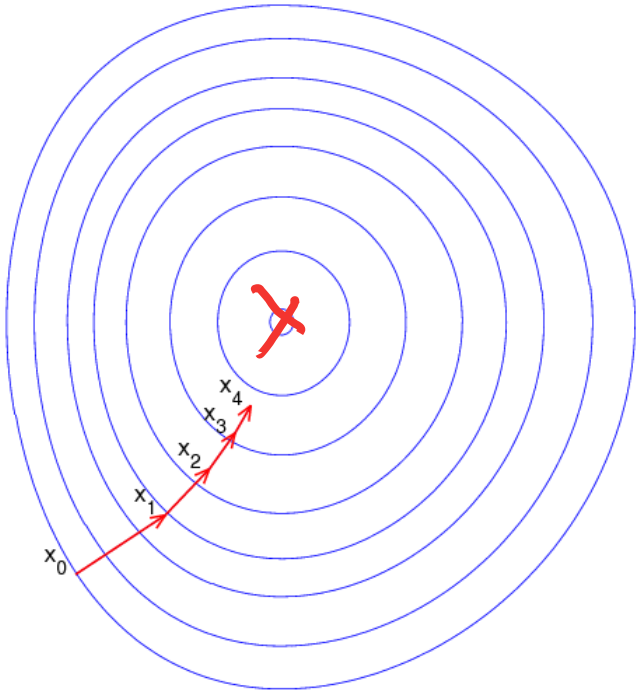
Stochastic gradient descent / Online Learning Algorithm

SGD



versus

GD



Stochastic gradient descent : More variations

- Single-sample:

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

- Mini-batch:

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

e.g. B = 15

Stochastic gradient descent (1)

- Very useful when training with massive datasets , e.g. not fit in main memory
- SGD can be used for offline training, by repeated cycling through the data
  - Each such pass over the whole data → an epoch !

- In offline case, often better to use mini-batch SGD
  - B=1 standard SGD
  - B=N standard batch GD
  - E.g. B=500

Stochastic gradient descent (2)

- Efficiency: Good approximation of Gradient:
  - Intuitively fairly good estimation of the gradient by looking at just a few examples
  - Carefully evaluating precise gradient using large set of examples is often a waste of time (because need to calculate the gradient of the next t any way)
  - Better to get a noisy estimate and move rapidly in the parameter space

- SGD is often less prone to stuck in shallow local minima
  - Because of the certain "noise",
  - popular for nonconvex optimization cases)

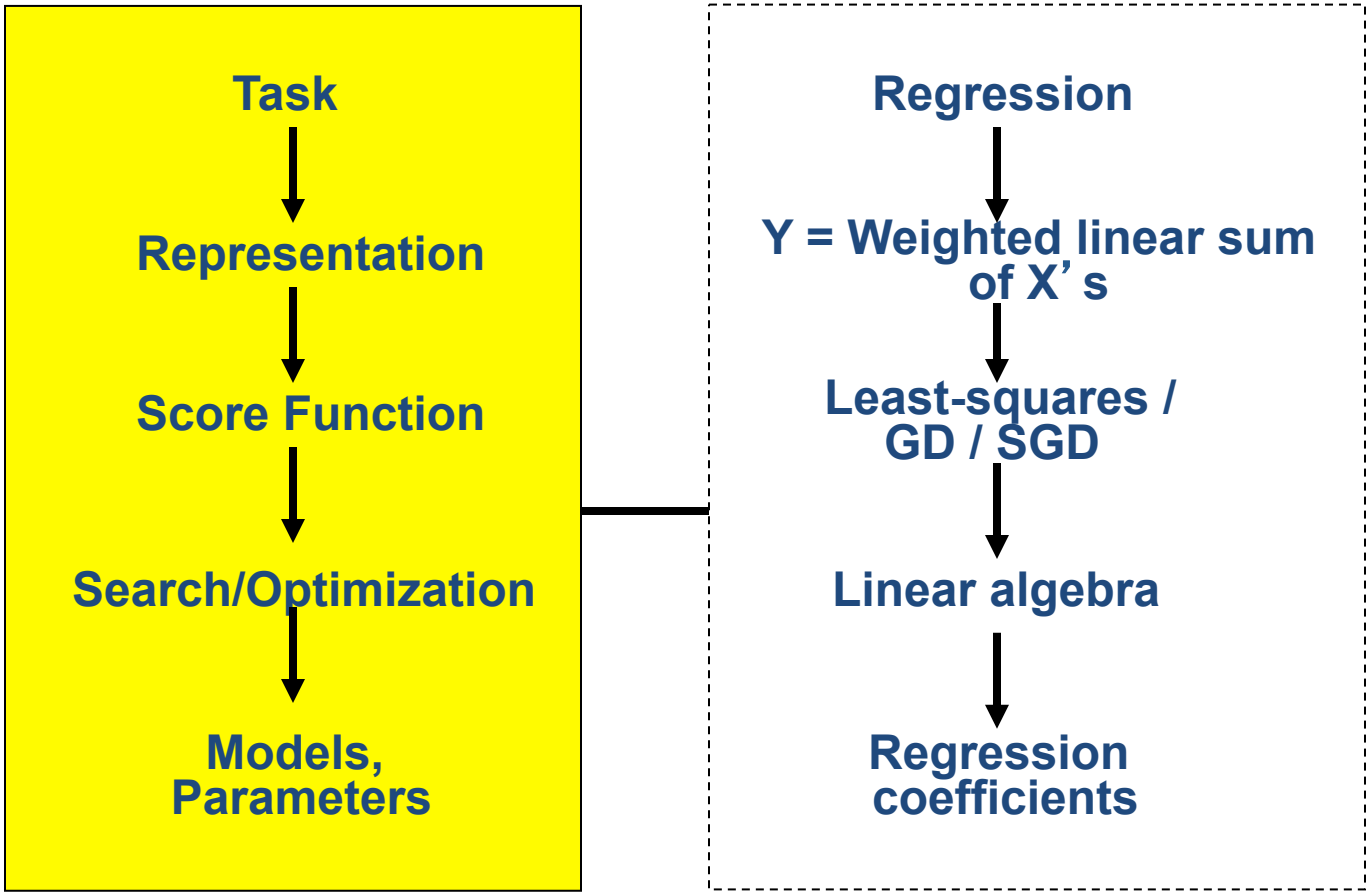
When to stop (S)GD ?

- Lots of stopping rules in the literature,
- There are advantages and disadvantages to each, depending on context
- E.g., a predetermined maximum number of iterations
- E.g., stop when the improvement drops below a threshold
- ….

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 $(X^T X)^{-1}$, 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 GD $\theta^{t+1} = \theta^t + \alpha (y_i - \mathbf{x}_i^T \theta^t)\mathbf{x}_i$
  - Pros: on-line, low per-step cost, fast convergence and perhaps less prone to local optimum
  - Cons: convergence to optimum not always guaranteed

(1) Multivariate Linear Regression



$$\hat{y} = f(x) = \theta^T x$$

References

- Big thanks to Prof. Eric Xing @ CMU for allowing me to reuse some of his slides

- Notes about Gradient Descent from Toussaint: 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

Extra: Computational Cost (Scalable?)

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

$X^T X$: $O(p^2 n)$

$(X^T X)^{-1}$: $O(p^3)$

$O(np^2 + p^3)$

when $n \gg p$, matrix multi: slower than inversion

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 GD or GD
  - Converging in a limiting sense
  - But more attractive in large practical problems
  - Caution is needed for deciding the learning rate

Extra: Convergence rate

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

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

If the learning rate parameter satisfy ➔

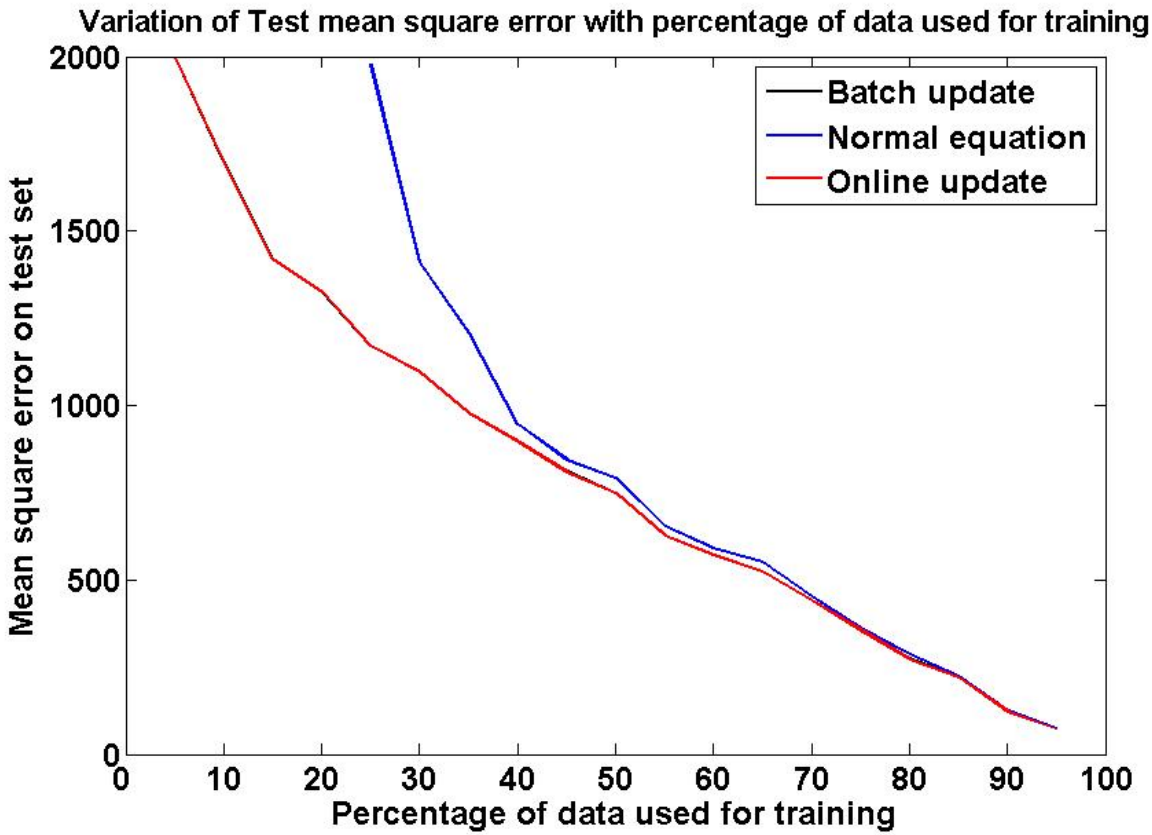
$$0 < \alpha < 2/\lambda_{\max}[X^T X]$$

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

$\alpha_0 = 0.05$

Dr. EricXing's tutorial slide

Extra: Performance vs. Training Size for an example



- The results from Batch GD and O (SGD) update are almost identical. So the plots coincide.
- The test MSE from the normal equation is more than that of BGD and O(SGD) 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.

Dr. EricXing's tutorial slide

Extra: Normal Equation Connecting to Newton's Method

Newton's method for optimization

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

GD: $\theta_{k+1} = \theta_k - \alpha g_k$

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

$p \times p$ $p \times 1$

$p \times 1$

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 !)

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

$$\Downarrow$$

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

$$\frac{\partial \hat{f}(\theta)}{\partial \theta} = 0 + g_k + \frac{2}{2} H_k \theta - \frac{2}{2} H_k \theta_k := 0$$

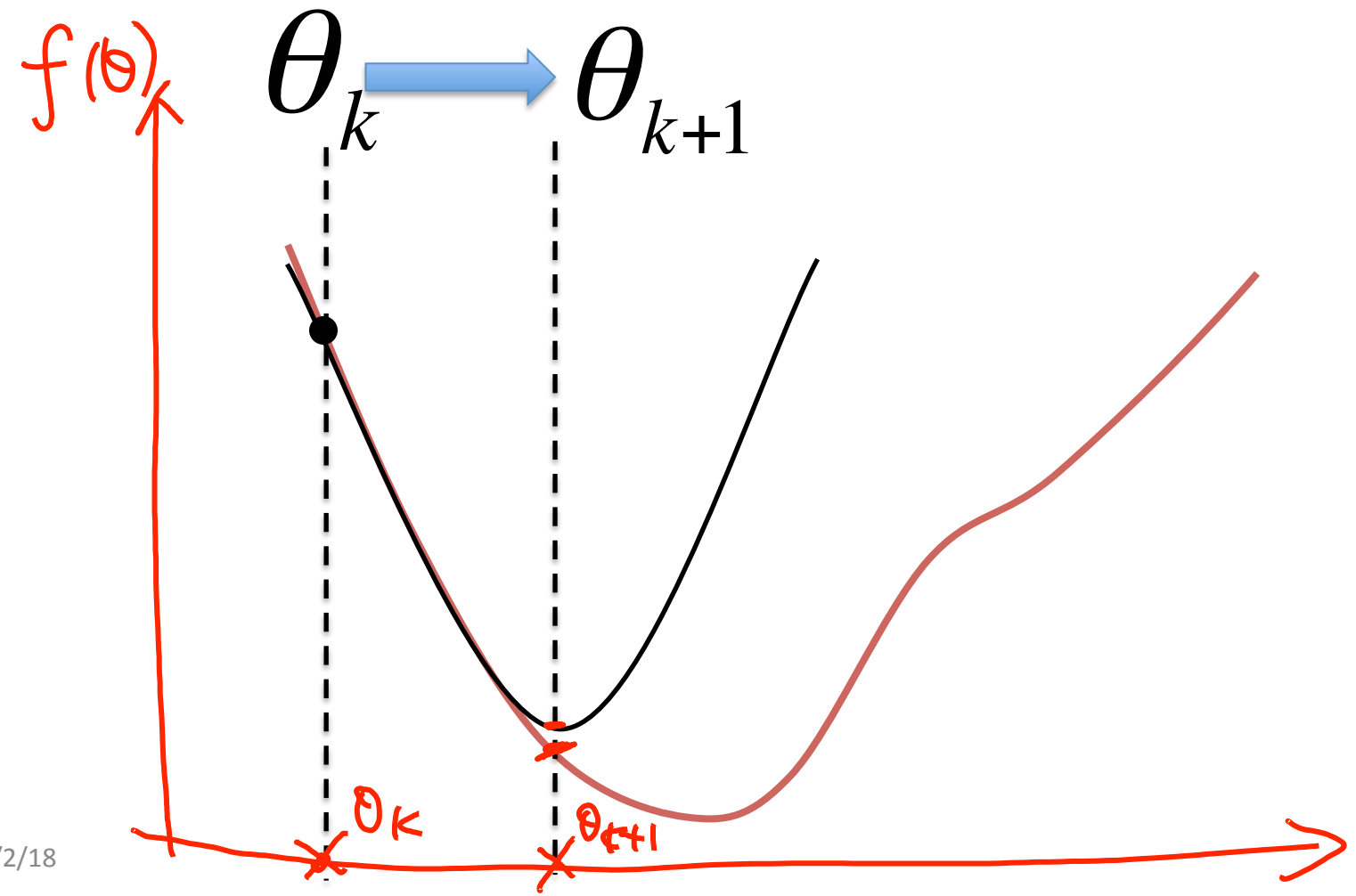
see p24 handout

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

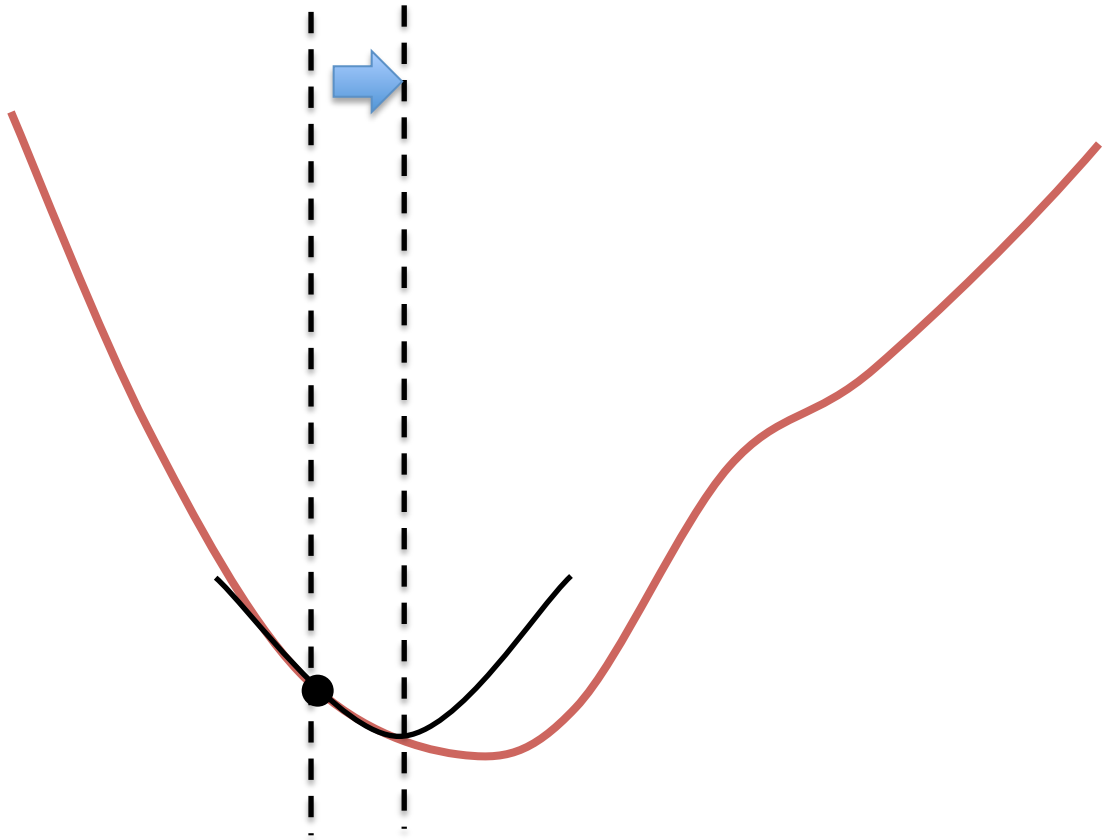
$$\Rightarrow \quad \theta = \theta_k - H_k^{-1} g_k$$

where $H_k \in R^{P \times P}$, $g_k \in R^P$

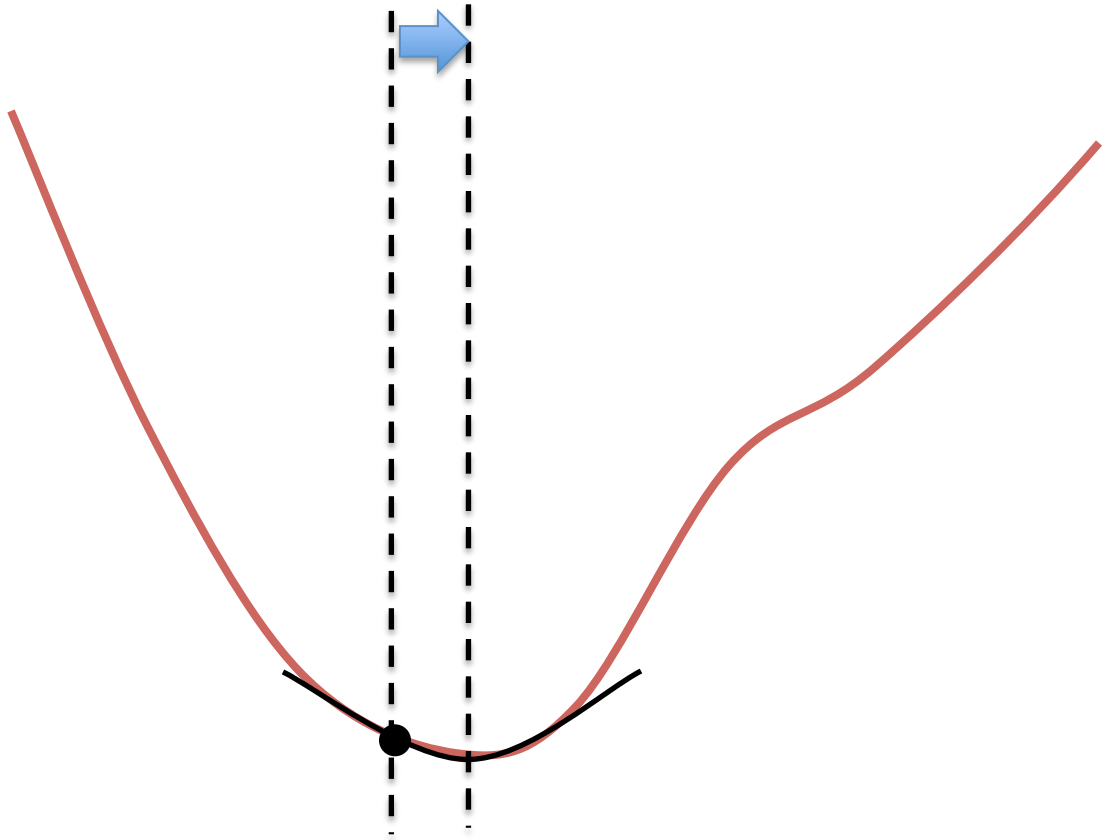
Newton's Method / second-order Taylor series approximation



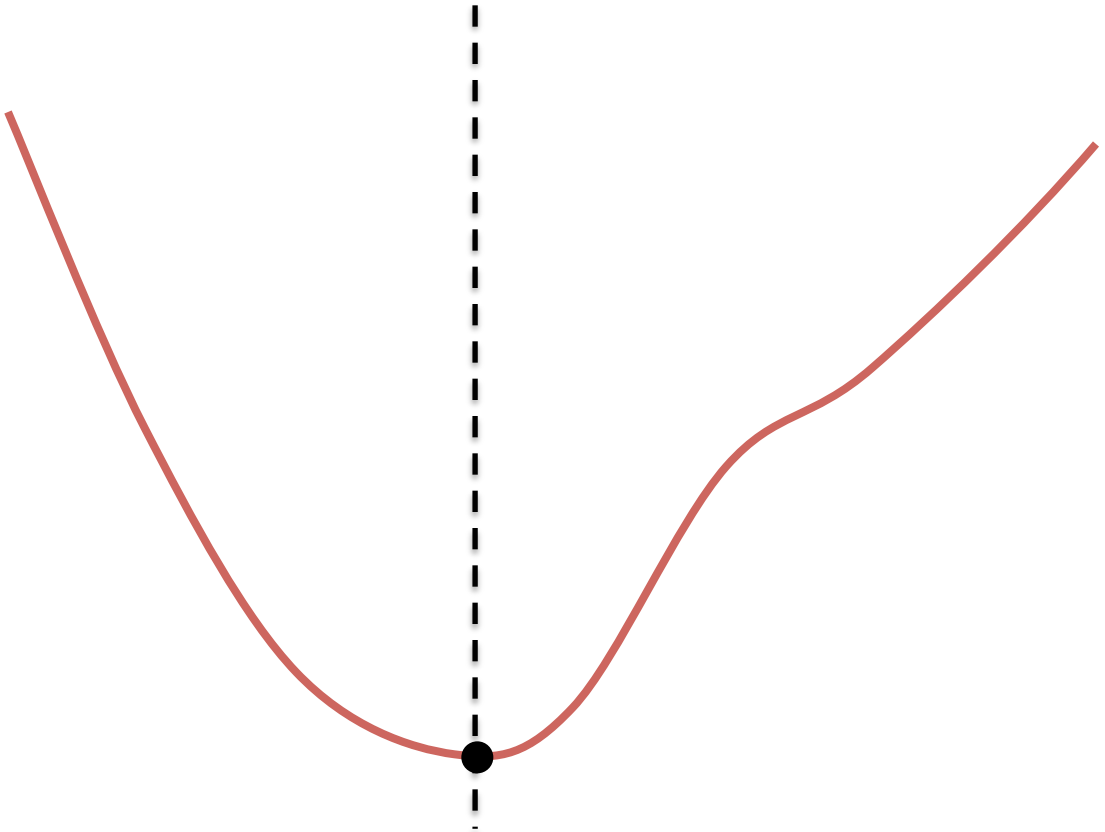
Newton's Method / second-order Taylor series approximation



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

Newton vs. GD for optimization

- Newton: a quadratic/second-order Taylor series approximation

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

$$\theta_{k+1} = \theta_k - \frac{1}{H(\theta_k)} g(\theta_k)$$

- GD: an approximation

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

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

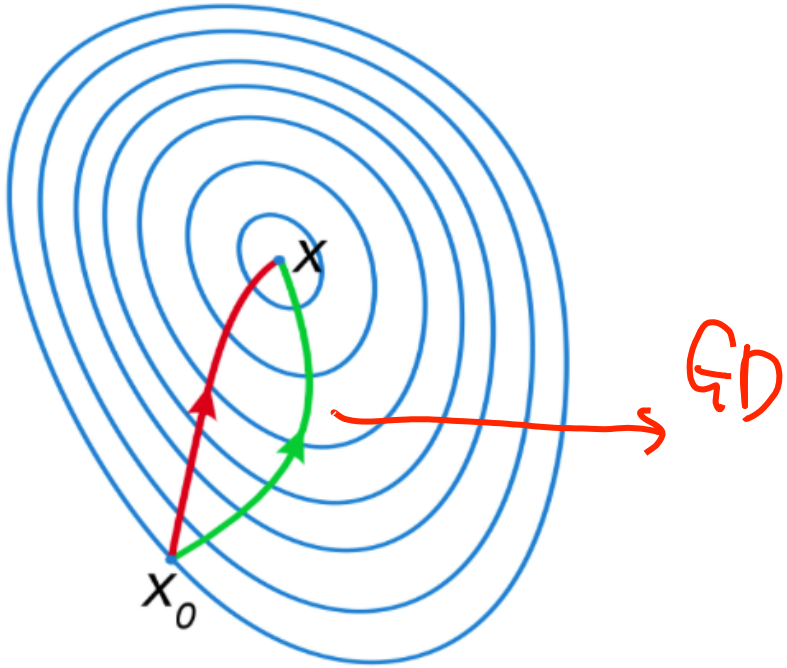
$$\theta_{k+1} = \theta_k - \alpha\, g(\theta_k)$$

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 …



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

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

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

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

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

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

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

WHY ??? Normal Eq?

Newton's method for Linear Regression