UVA CS 6316: Machine Learning

Lecture 4: More optimization for Linear Regression

Dr. Yanjun Qi

University of Virginia

Department of Computer Science

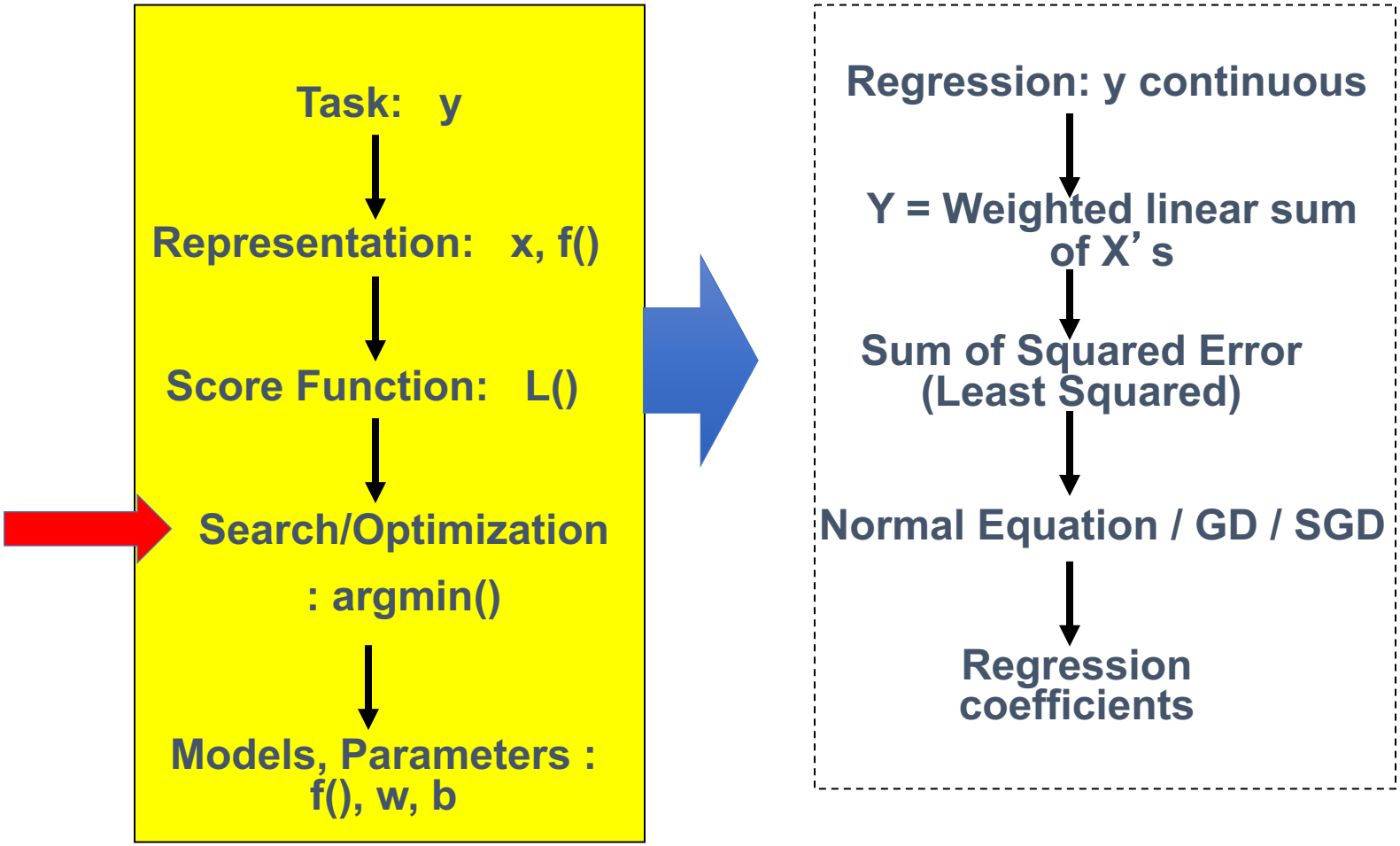
Course Content Plan ➔
Six major sections of this course

- ❑ Regression (supervised) — Y is a continuous
- ❑ Classification (supervised) — Y is a discrete
- ❑ Unsupervised models — NO Y
- ❑ Learning theory — About f()

- ❑ Graphical models — About interactions among X1,… Xp

- ❑ Reinforcement Learning — Learn program to Interact with its environment

Today: Multivariate Linear Regression in a Nutshell



Learning Regression Models (supervised)

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

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:

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

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

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

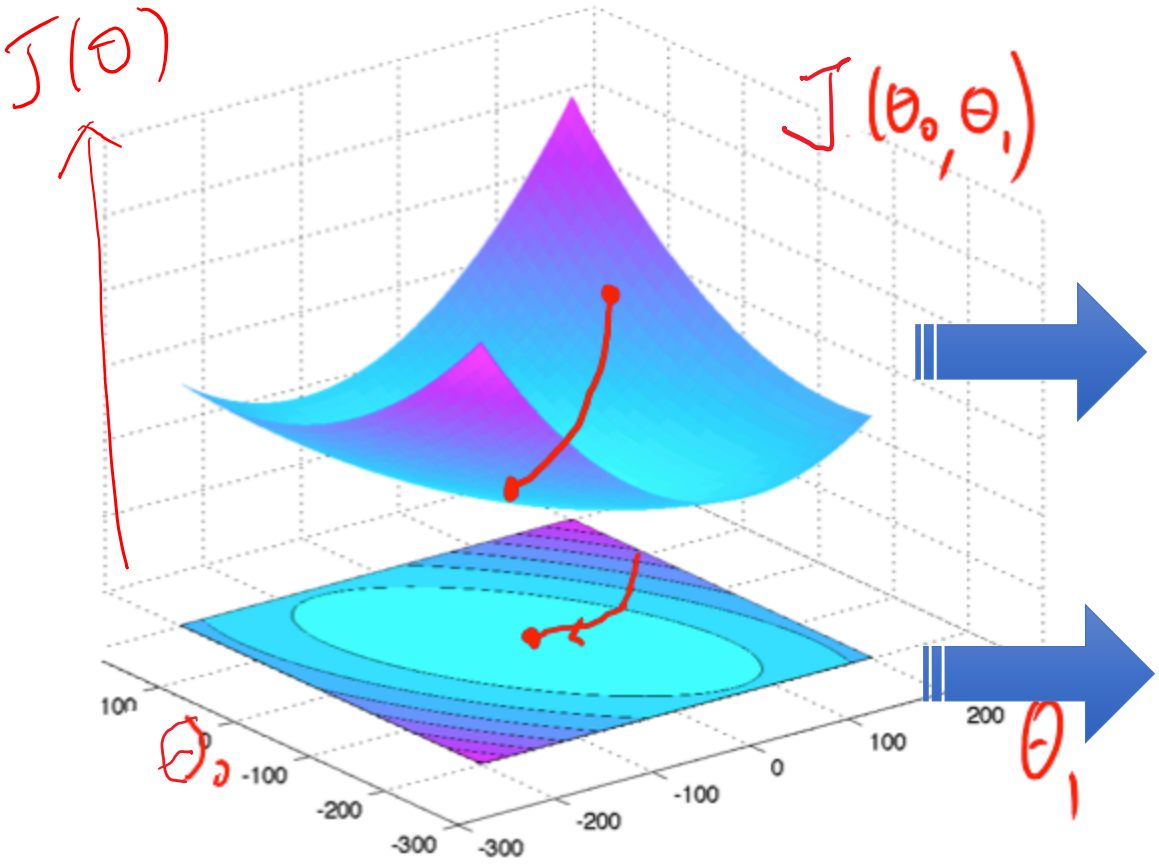
↓

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

Today

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

Two ways of Illustrating an Objective Function (e.g. 2D case)



Gradient vector points to the direction of greatest rate of increase of the objective function and its magnitude is the slope of the surface graph in that direction..

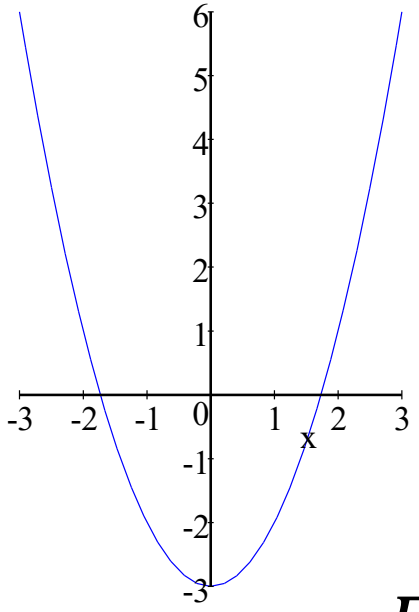
Review: Definitions of gradient (in L2-note)

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

$$\nabla_{\boldsymbol{x}} F(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial F(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial F(\boldsymbol{x})}{\partial x_2} \\ \cdots \\ \frac{\partial F(\boldsymbol{x})}{\partial x_p} \end{bmatrix} \in \mathbb{R}^p \quad \text{if} \quad \boldsymbol{x} \in \mathbb{R}^p$$

A vector whose entries respectively contain the p partial derivatives

Review: Definitions of derivative (1D case)



Review: Derivative of a Quadratic Function

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

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

The derivative is often described as the "instantaneous rate of change",
➔ the ratio of the instantaneous change in F(x) to in x

WHY ? Optimize through Gradient Descent (iterative) Algorithms

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

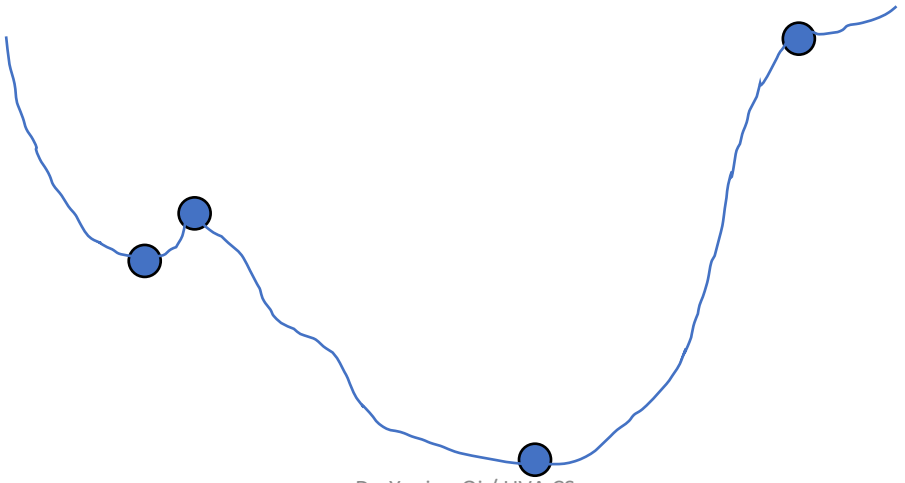
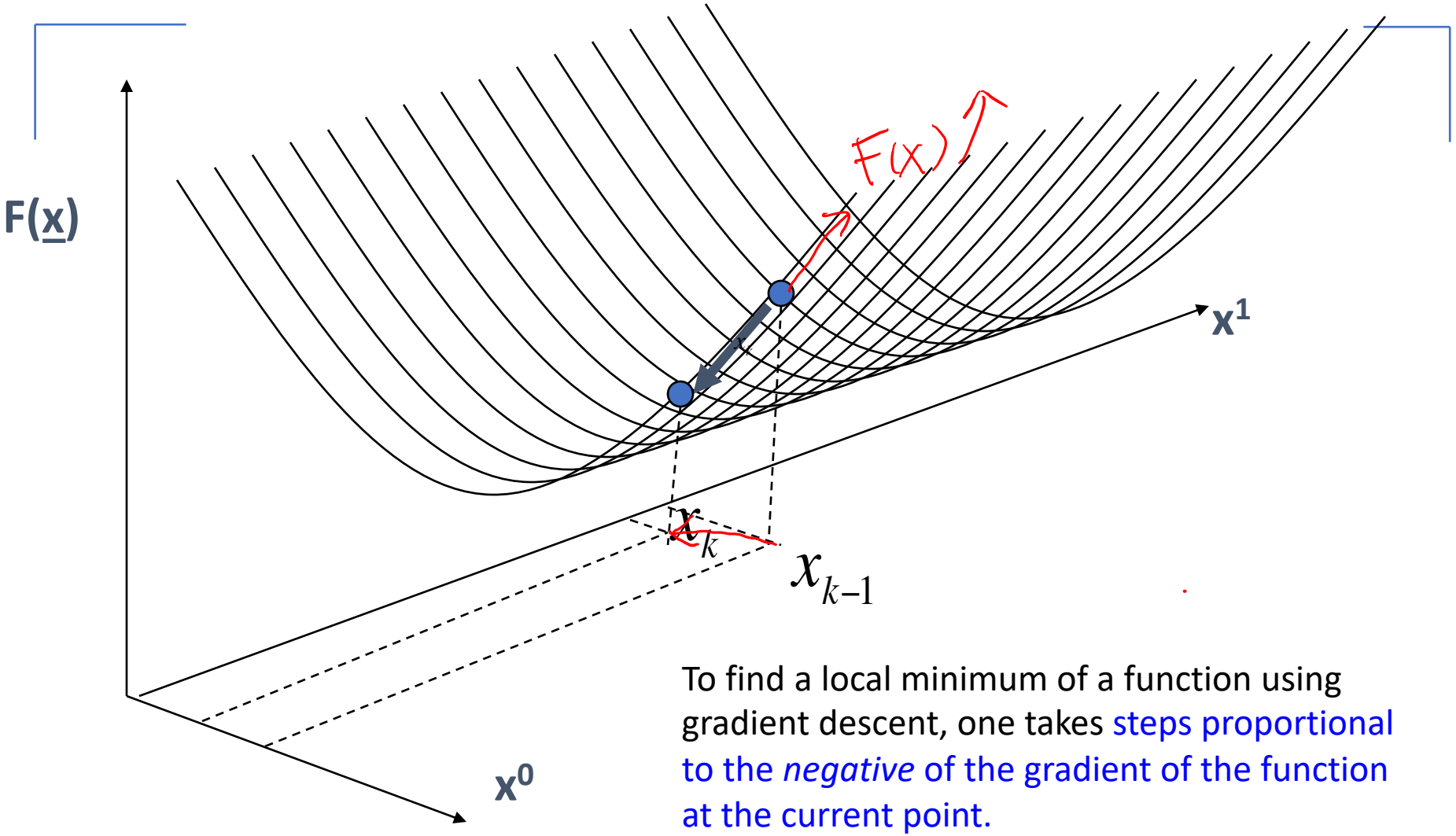
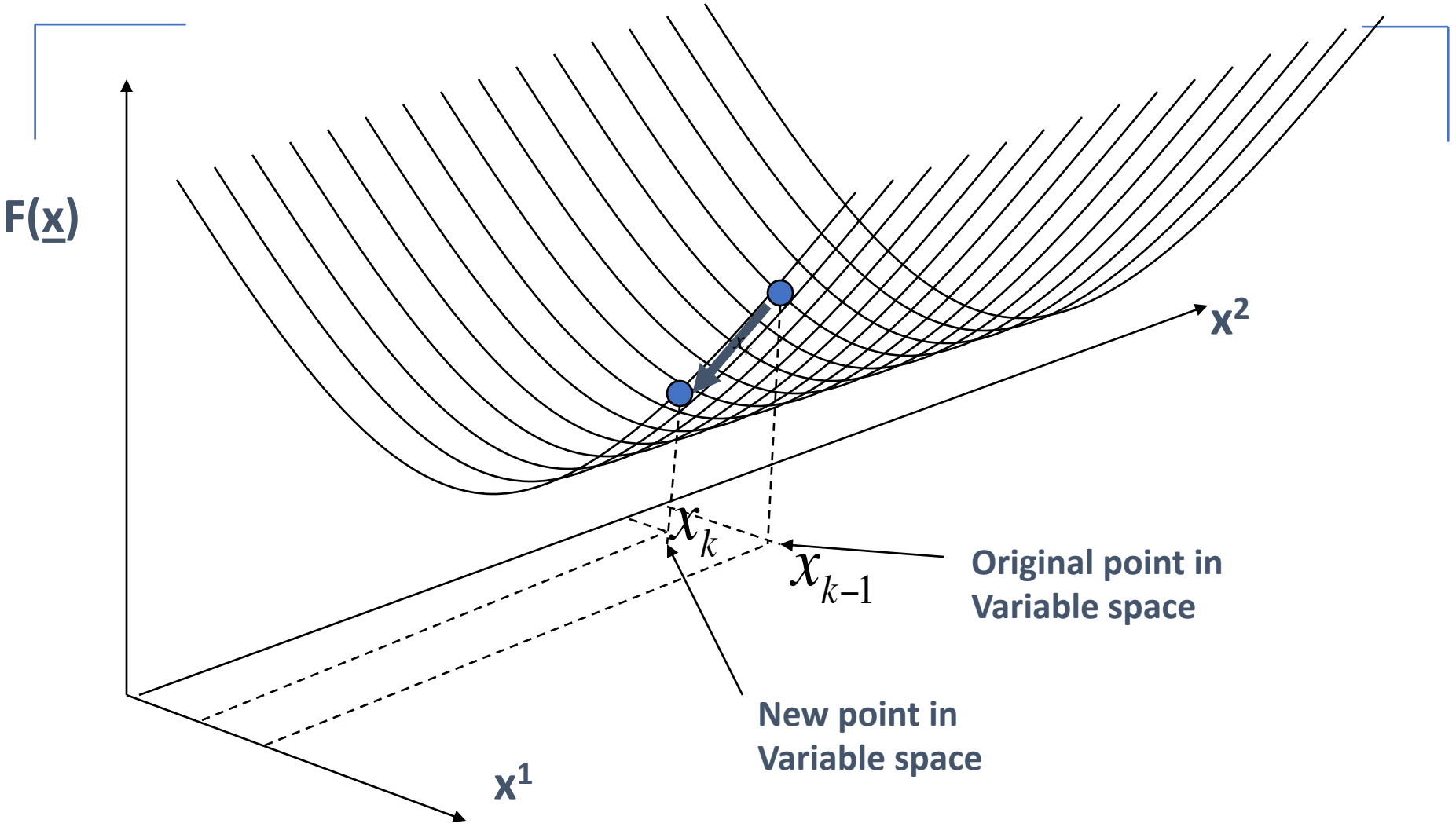


Illustration of Gradient Descent (2D case)

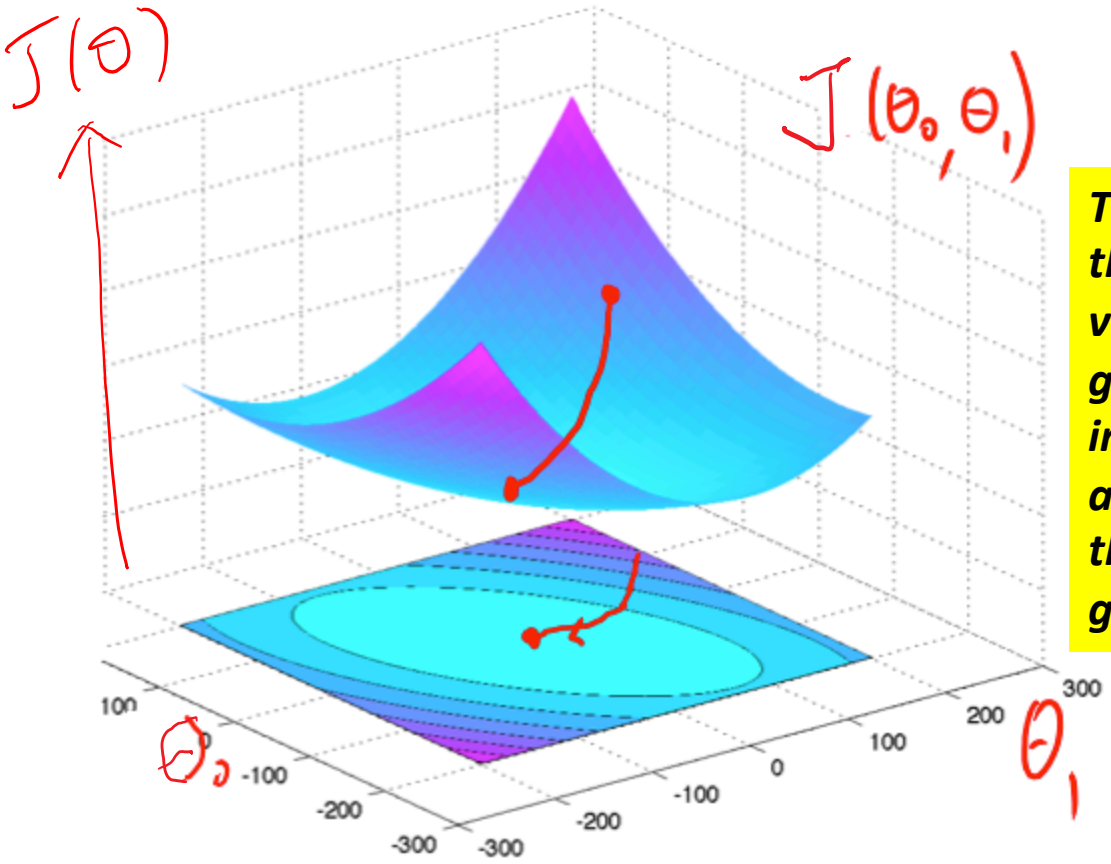


To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient of the function at the current point.

Illustration of Gradient Descent (2D case)



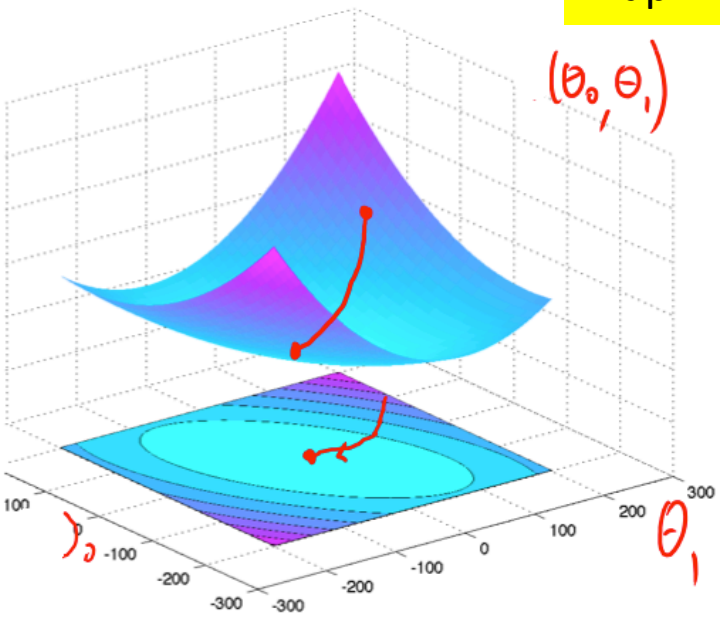
Two ways of Illustrating the Objective Function and Gradient Descent (e.g. , 2D case)

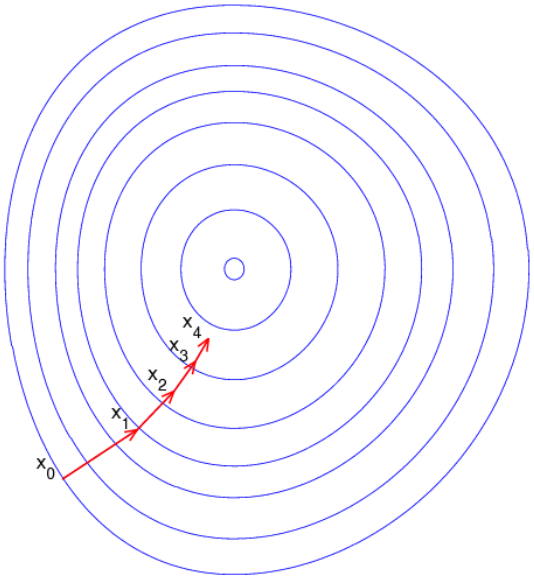


The gradient points in the direction (in the variable space) of the greatest rate of increase of the function and its magnitude is the slope of the surface graph in that direction

Surface map

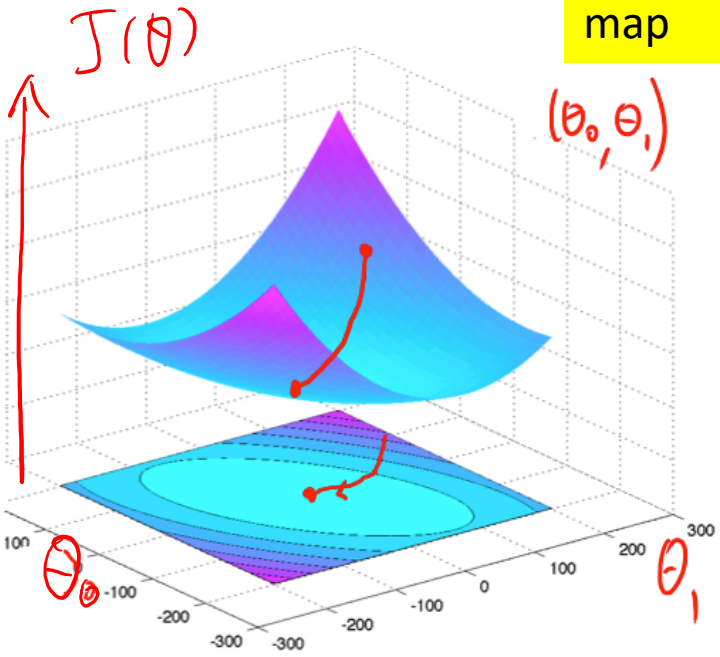
Contour map

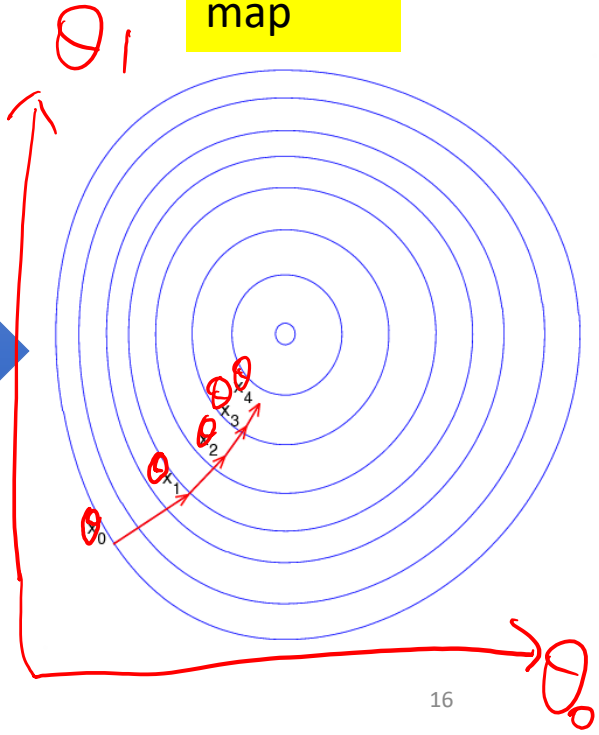




Surface map

Contour map



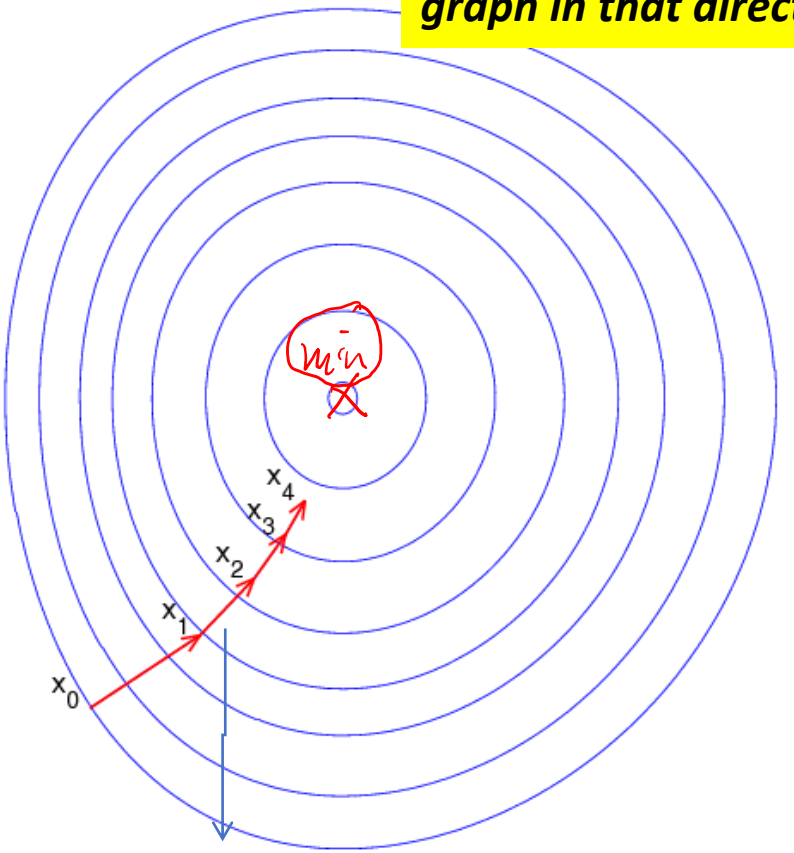


Gradient Descent (Steepest Descent) – contour map view

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.



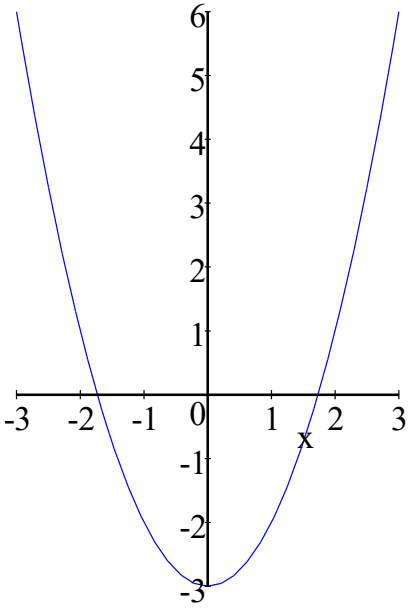
$$-\nabla_x F(x_{k-1})$$

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

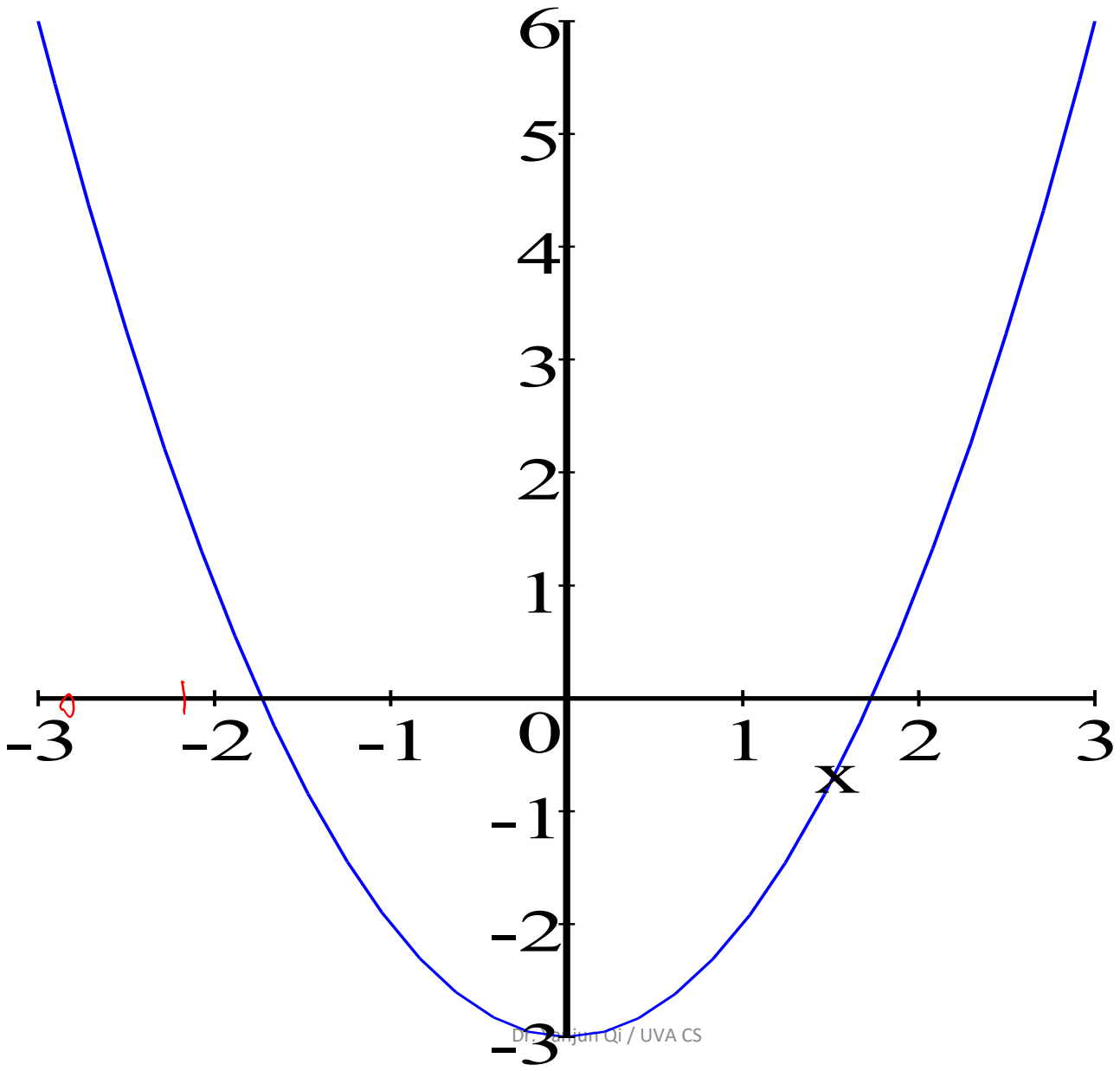
$$F(x) = x^2 - 3$$

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

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

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

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



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

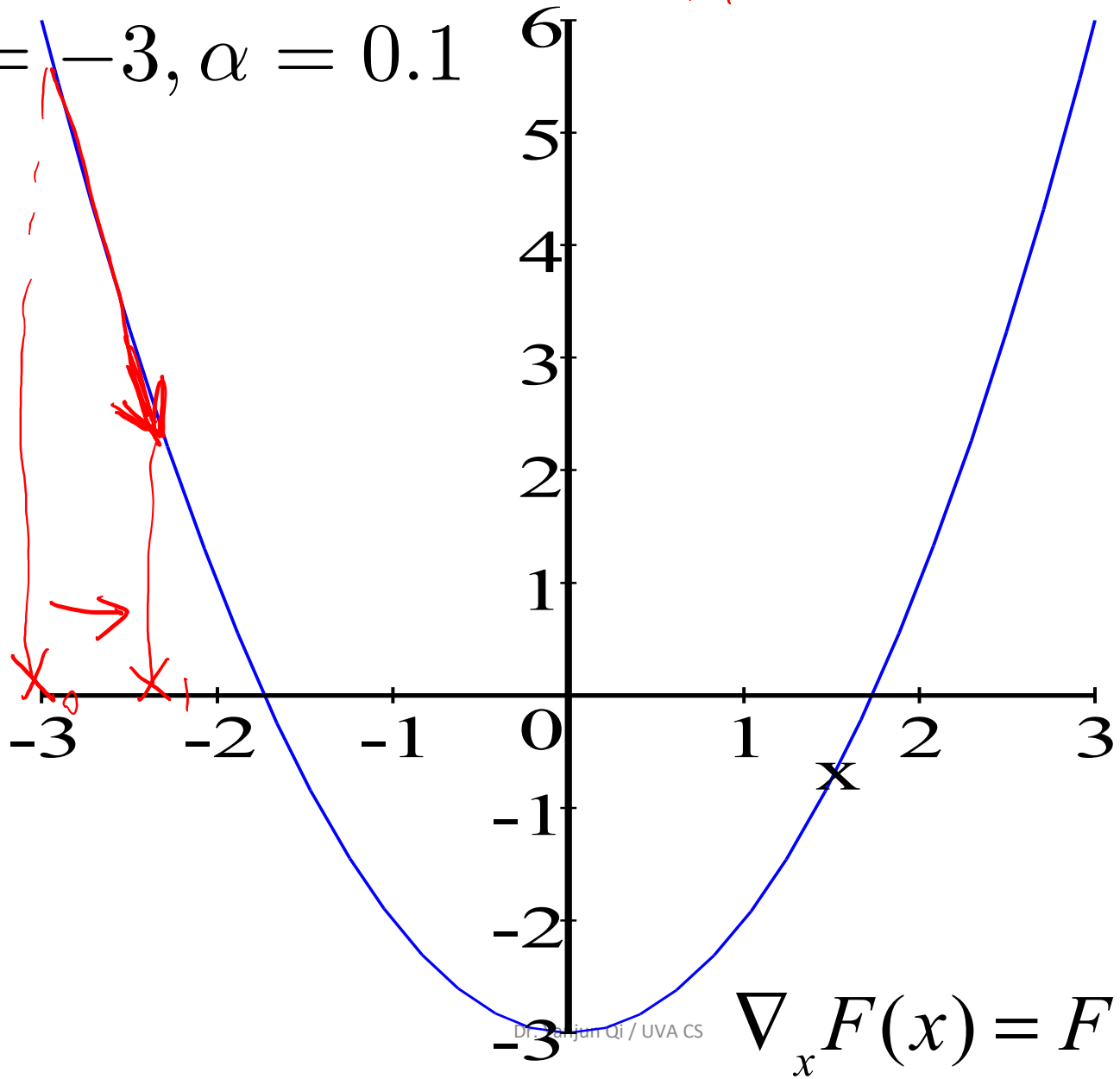
$$x_0 = -3, \alpha = 0.1$$

$X_1 = X_0 - \alpha \nabla F(X_0)$

$= -3$

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

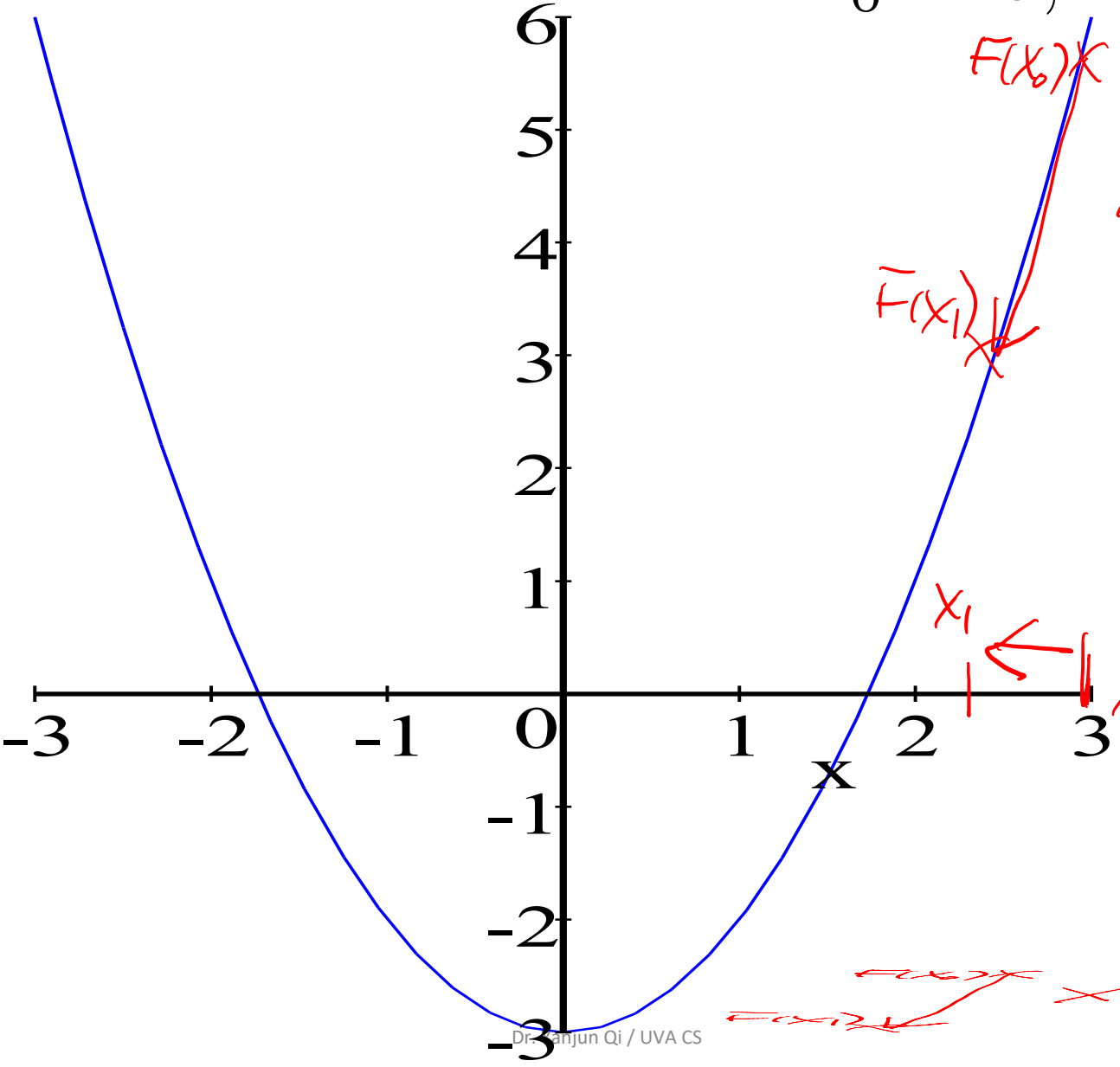
$= -2.4$



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

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

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



$F(x_0)$

$X_1 = 2.4$

$F(x_1)$

x_1

x_0

$x_1 = 2.4$

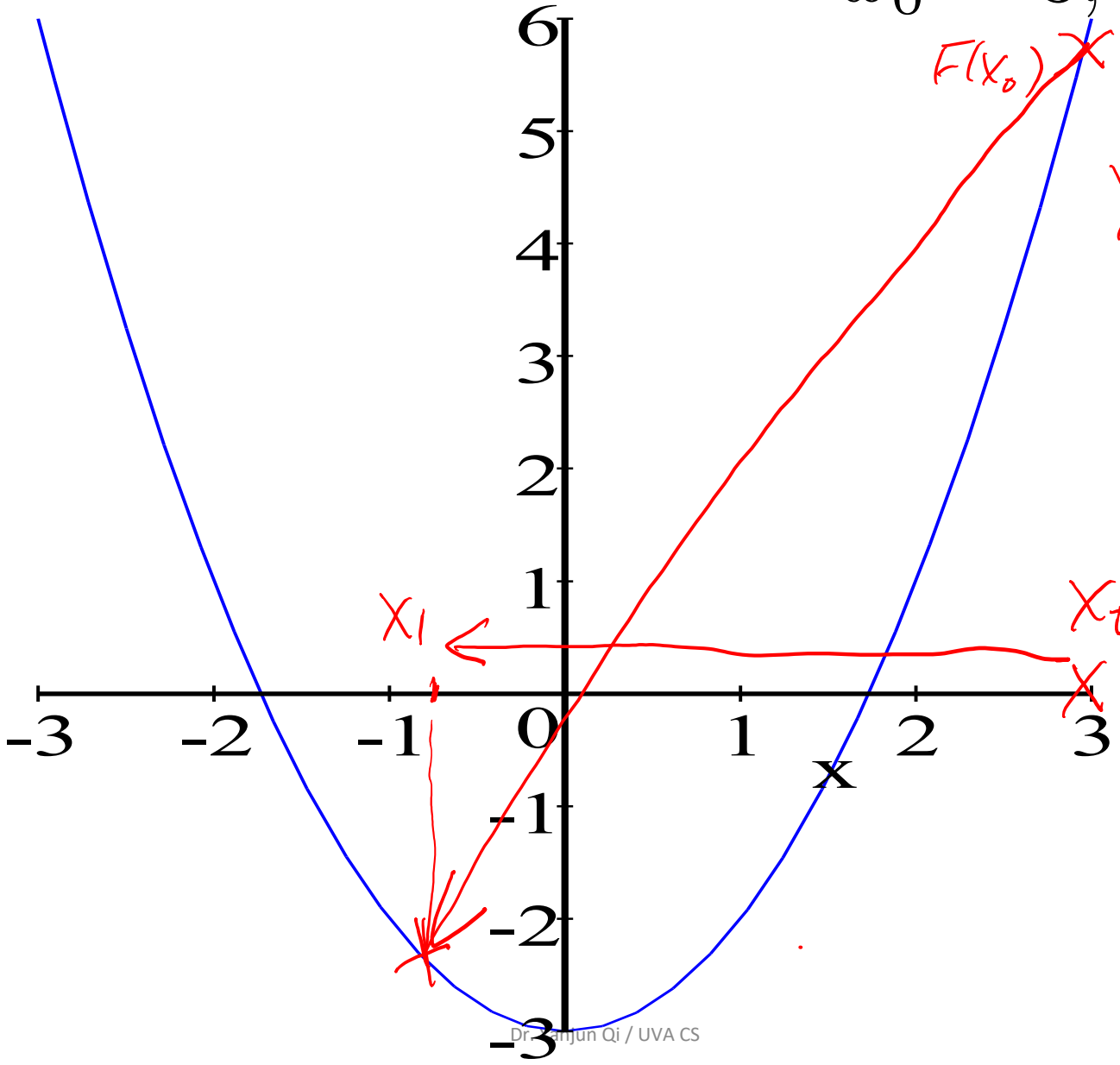
$x_1 \leftarrow x_0$

Gradient Descent (Iteratively Optimize)

- Learning Rate Matters
- Starting point matters
- Objective function matters

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

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

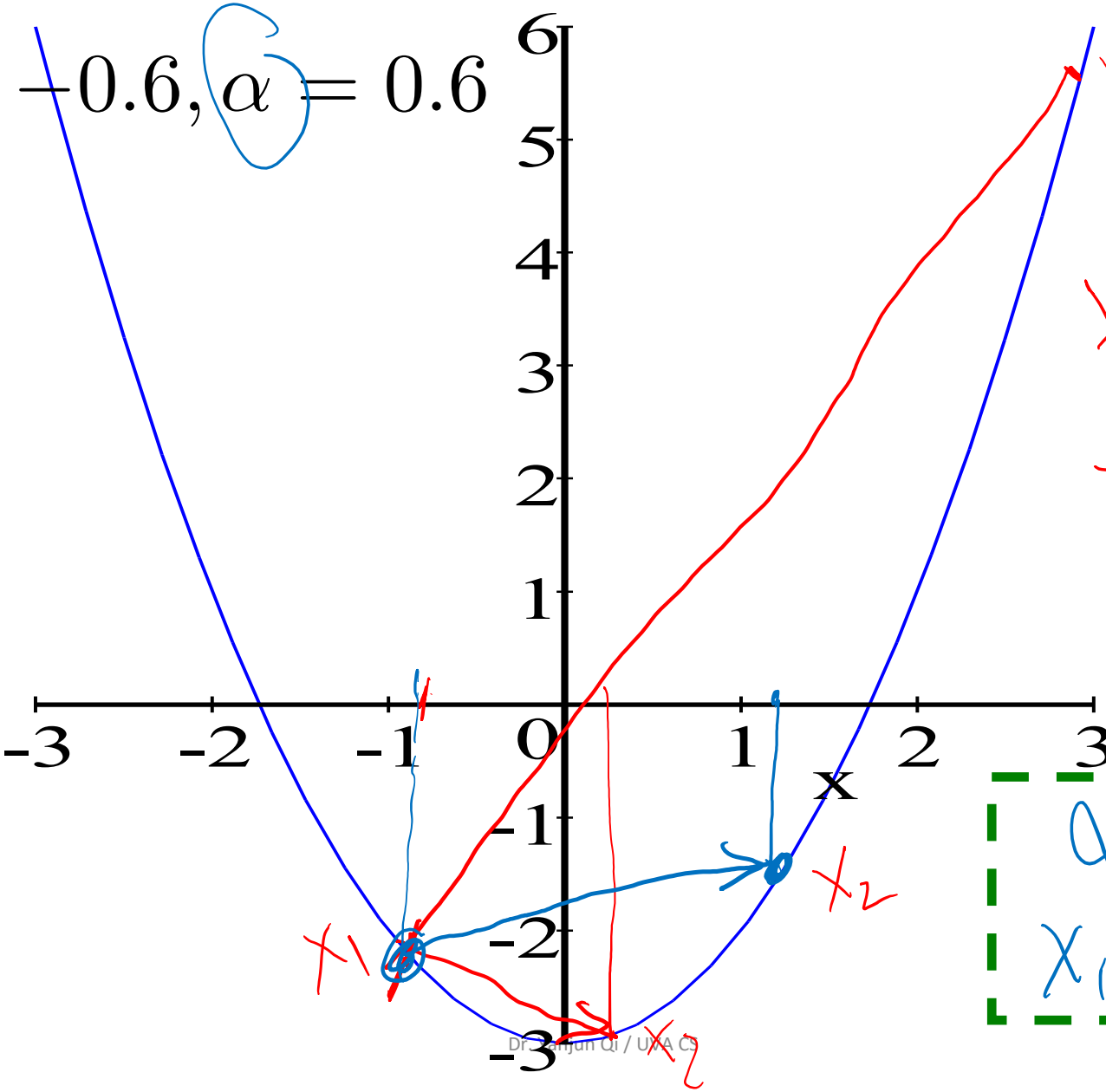


Gradient Descent (Iteratively Optimize)

- Learning Rate Matters
- Starting point matters
- Objective function matters

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

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



$x_1 = -0.6$

$+2 \times 0.6 \times 0.6$

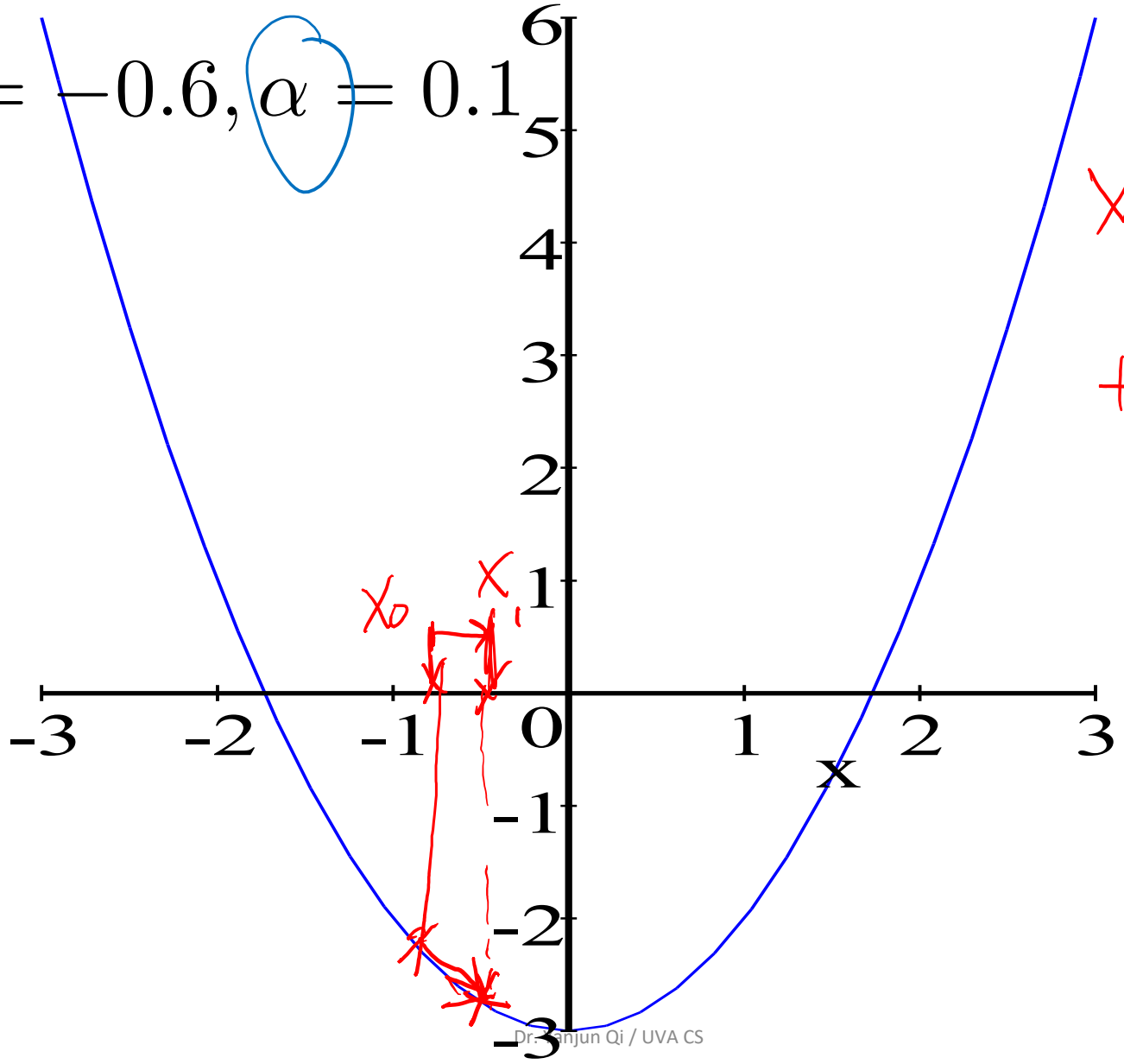
$= 0.12$

$\alpha = 1.5$

$x_1 = 1.2$

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

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



$X_1 = -0.6$

$+2 \times 0.1 \times 0.6$

$= -0.48$

Gradient Descent (Iteratively Optimize)

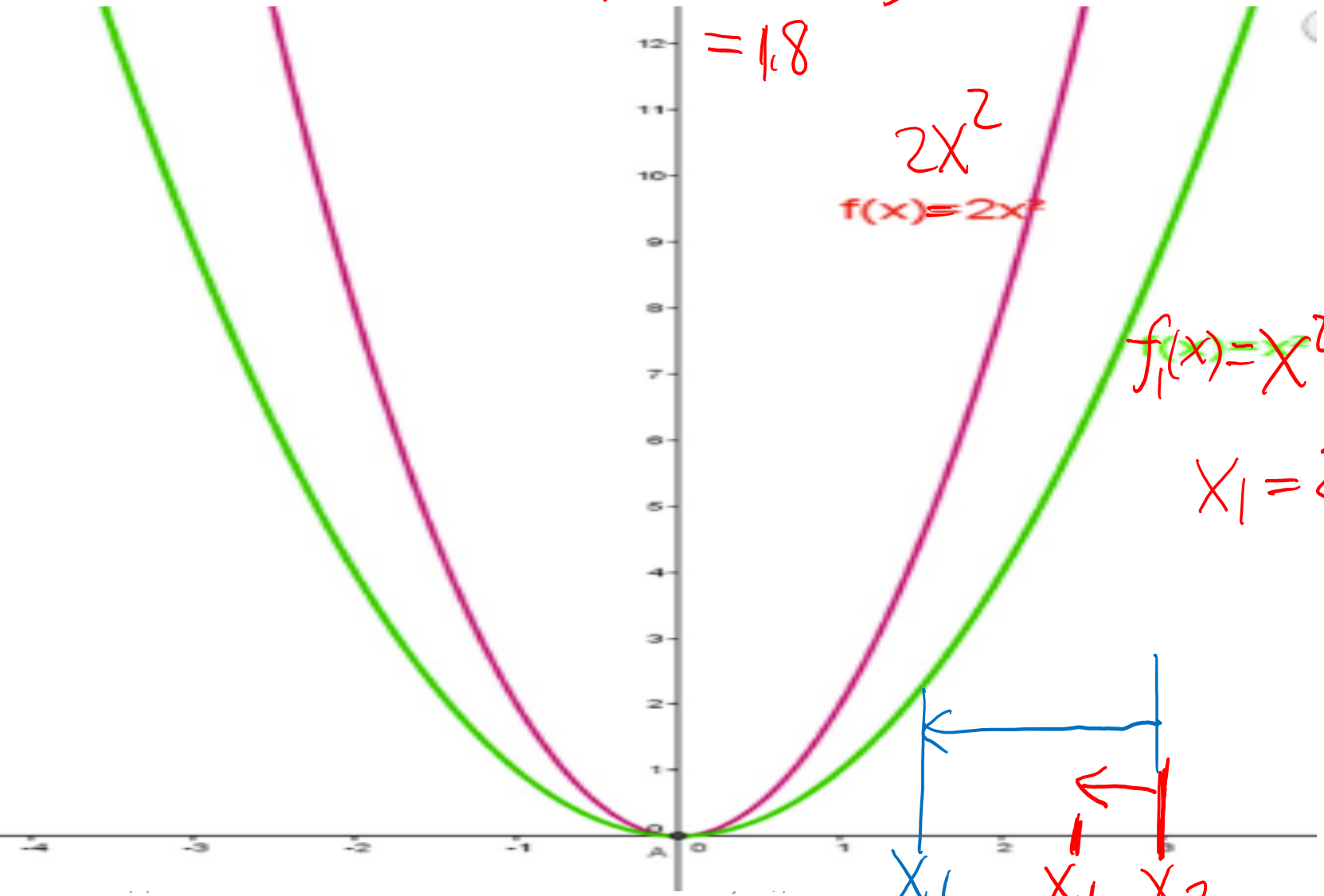
- Learning Rate Matters
- Starting point matters
- Objective function matters

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

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

$X_1 = 3 - 4 \times 0.1 \times 3$

$= 1.8$

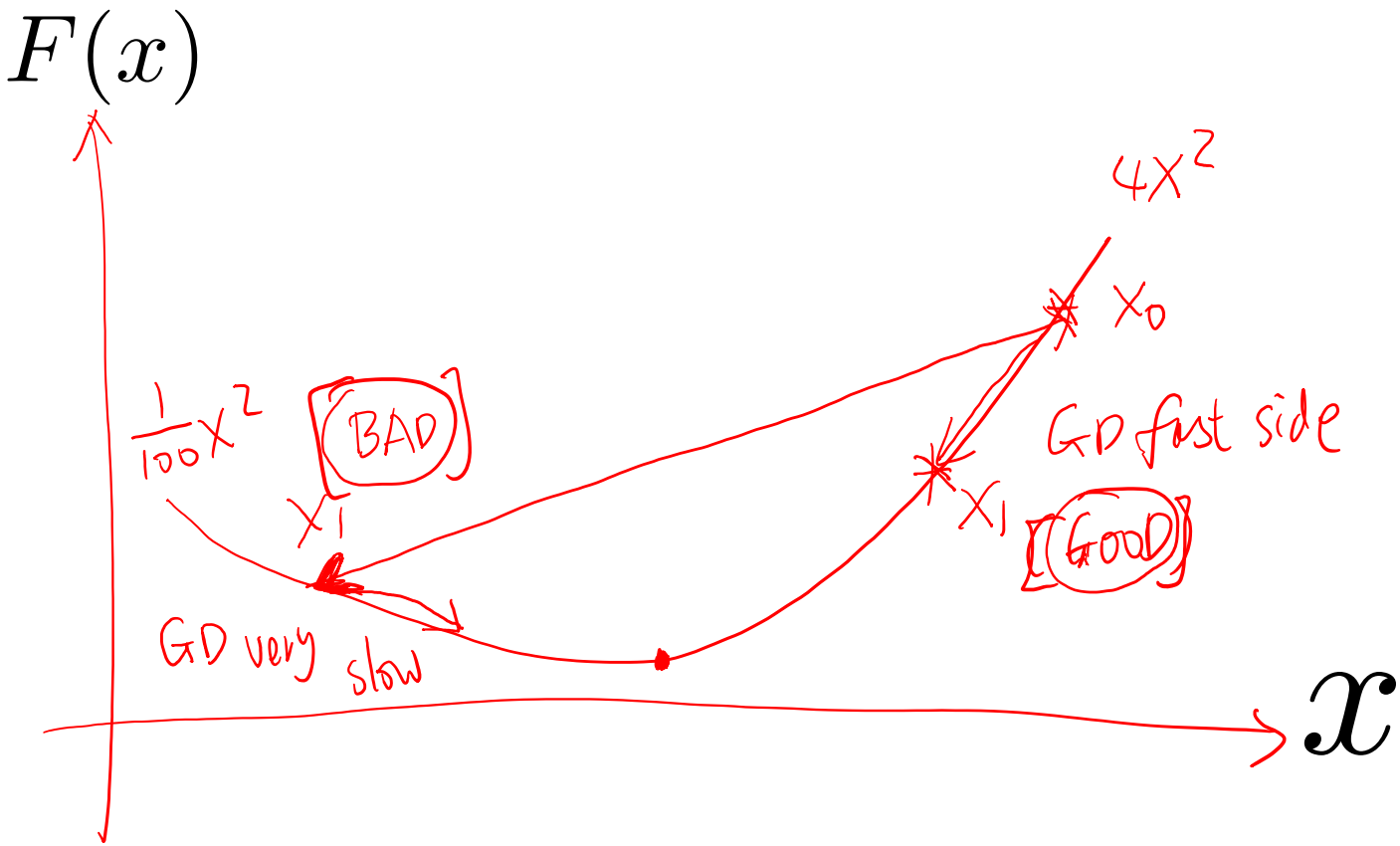


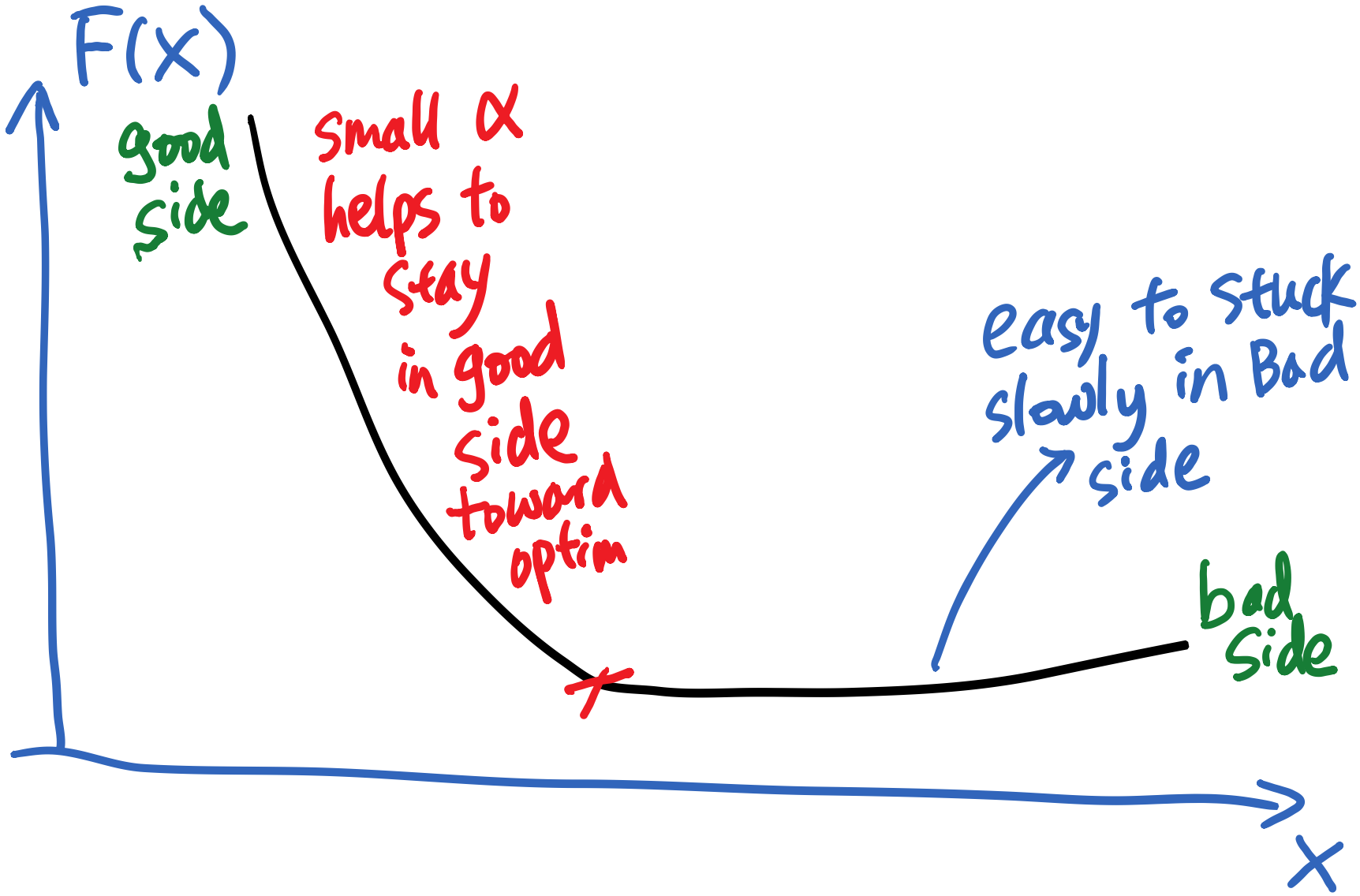
X_t	α	X_(t+1)	F(x)
-3	0.1	-2.4	x^2
3	0.1	2.4	x^2
3 -0.6	0.6 0.6	-0.6 0.12	x^2
-0.6	0.1	-0.48	x^2
3	0.1	1.8	$2X^2$

α

① small

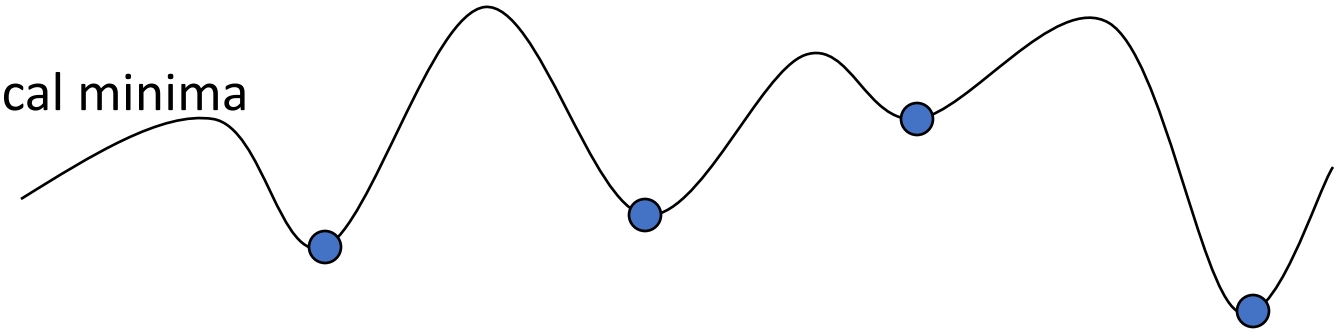
② smaller as t ↑


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


F(x)
good side
small α helps to stay in good side toword optim
easy to stuck slowly in Bad side
bad side
X

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 Least Square LR

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

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

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

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

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

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

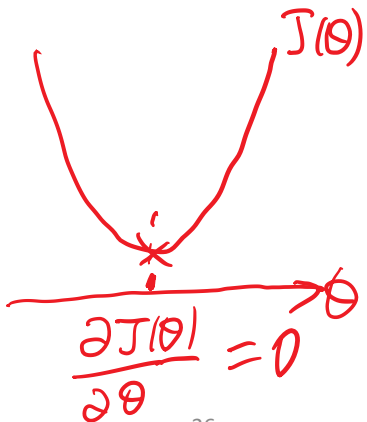
Since $\theta^T X^T y = y^T X\theta$

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

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

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

1d case



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

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

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

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

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

$$\nabla_\theta J(\theta)$$

$$= X^T X\theta - X^T \vec{y}$$

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

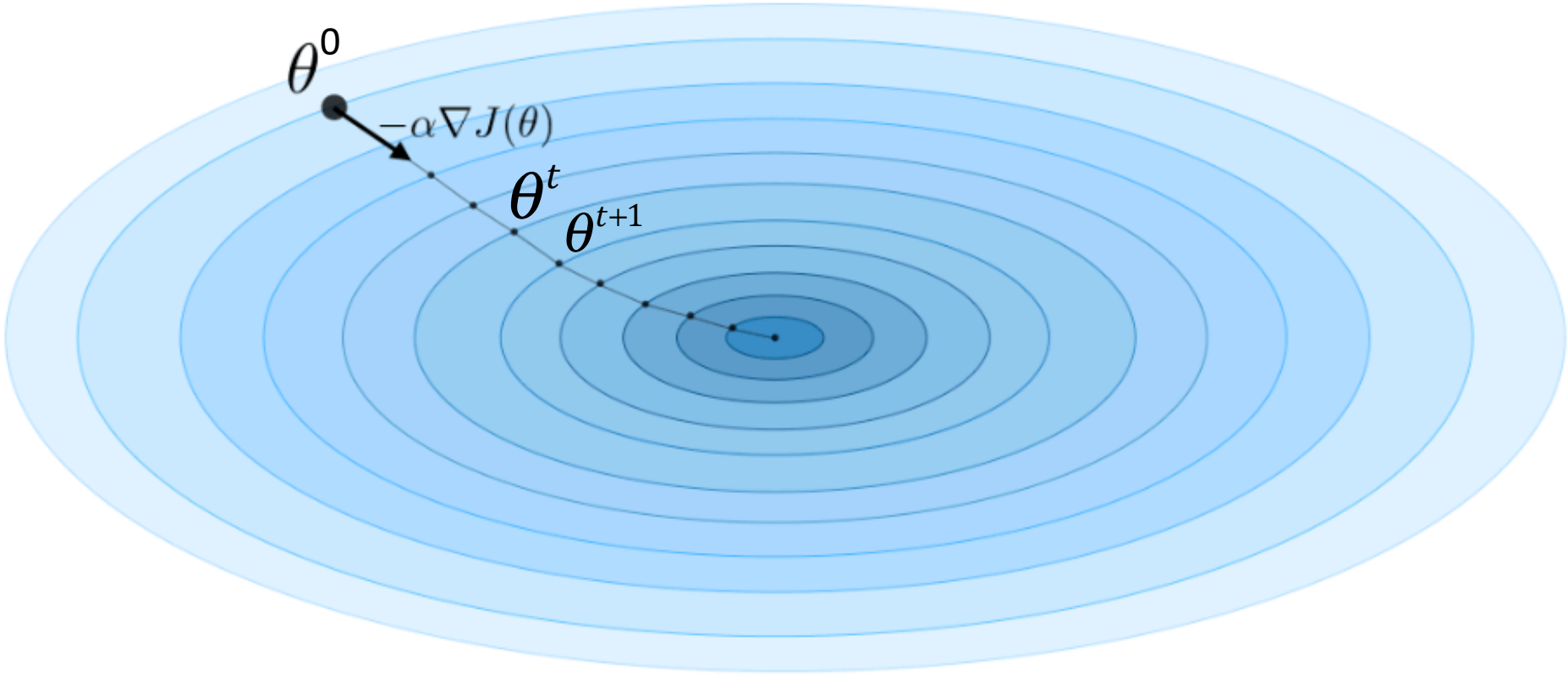
LR with batch GD

- A Batch **gradient descent** algorithm:

$$\theta^{t+1} = \theta^t - \alpha \nabla_\theta J(\theta^t)$$
$$= \theta^t + \alpha X^T (\vec{y} - X\theta^t)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}$$

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



$$\theta^{t+1} = \theta^t - \alpha\nabla_\theta J(\theta^t)$$

$$= \theta^t + \alpha X^T(\vec{y} - X\theta^t)$$

$$\theta^{t+1} = \theta^t - \alpha \nabla_\theta J(\theta^t)$$

$$= \theta^t + \alpha X^T (\vec{y} - X\theta^t)$$

$$\vec{\theta}^{t+1} = \vec{\theta}^t + \alpha X^T (\vec{y} - X\vec{\theta}^t)$$

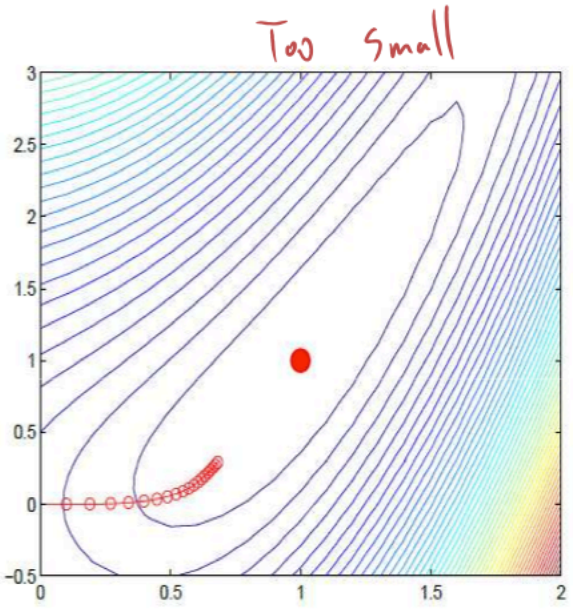
$p \times 1 \quad p \times 1 \quad 1 \times 1 \quad p \times n \quad n \times 1 \quad n \times p \quad p \times 1$

$n \times 1$

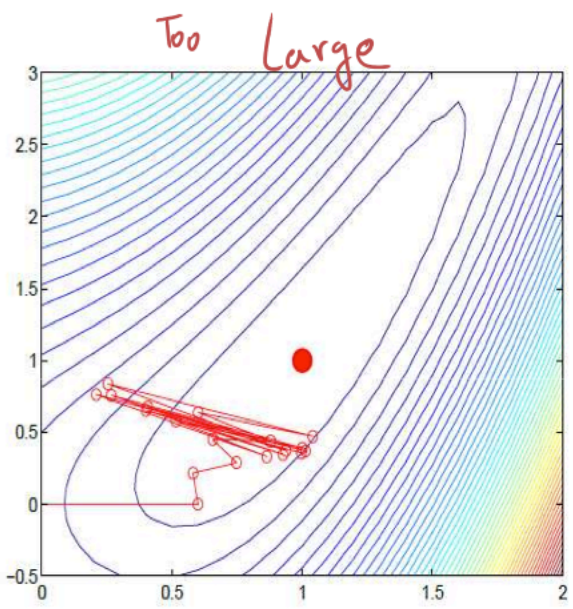
$n \times 1$

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

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 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 and reformulate:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}$$

$$\theta^{t+1} = \theta^t - \alpha\nabla_\theta J(\theta^t)$$

$$= \theta^t + \alpha X^T(\vec{y} - X\theta^t)$$

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

LR with Stochastic GD ➔

- Batch GD rule:

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

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

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

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

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

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

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

$n \times p$ $\quad p \times 1$

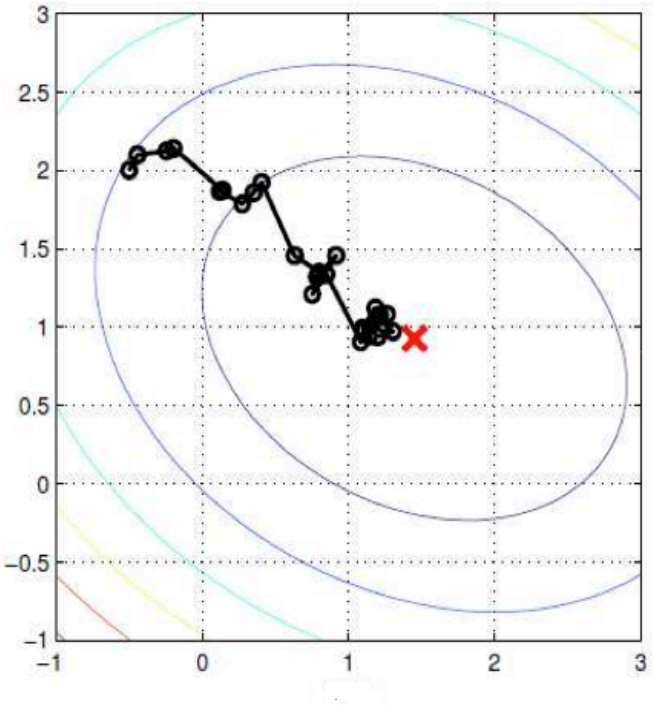
$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1^T & -- \\ -- & \mathbf{x}_2^T & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n^T & -- \end{bmatrix}$$

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

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

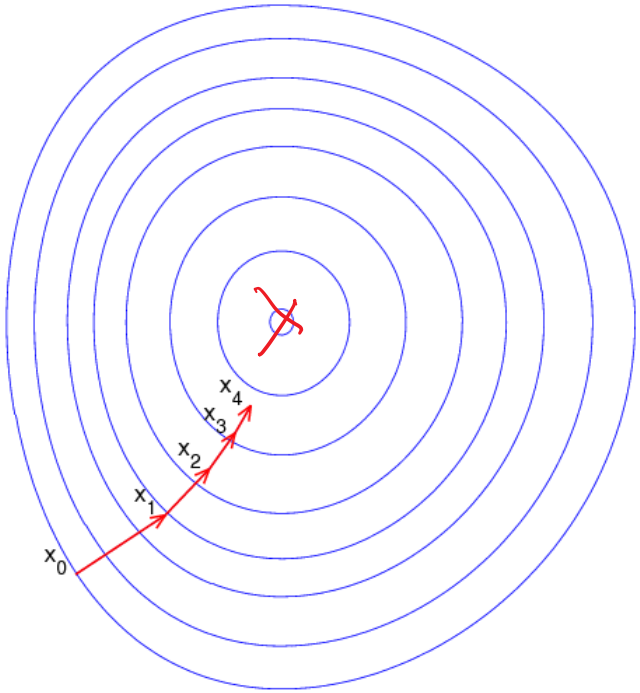
Stochastic gradient descent / Online Learning Algorithm

SGD



versus

GD



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 (1)

- Very useful when training with massive datasets , e.g. not fit in main memory
- Very useful when training data arrives online (e.g. streaming)..
- SGD can be used for offline training, by repeated cycling through the whole 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=50

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

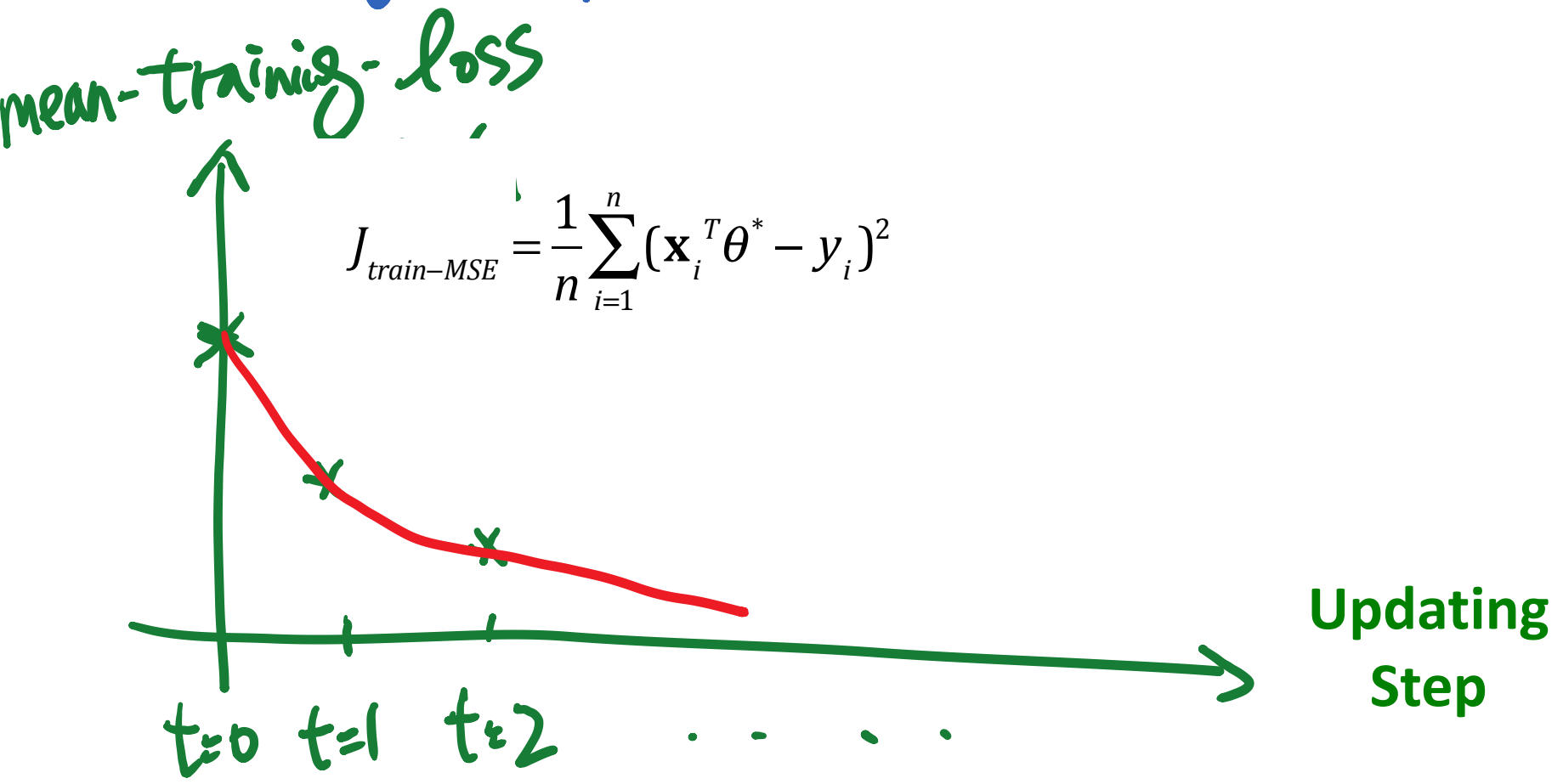
B

Varying the value B In

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

1	$1 < B < n$	n
SGD	miniB-SGD	GD
very noisy GD update	a bit noisy GD update	precise GD update
low memory cost	middle memory cost	high memory cost
O(one gradient cacl cost)	if multi-parallel B threads O(one gradient cal cost	very costly gradient calculation

One good plot for GD/SGD



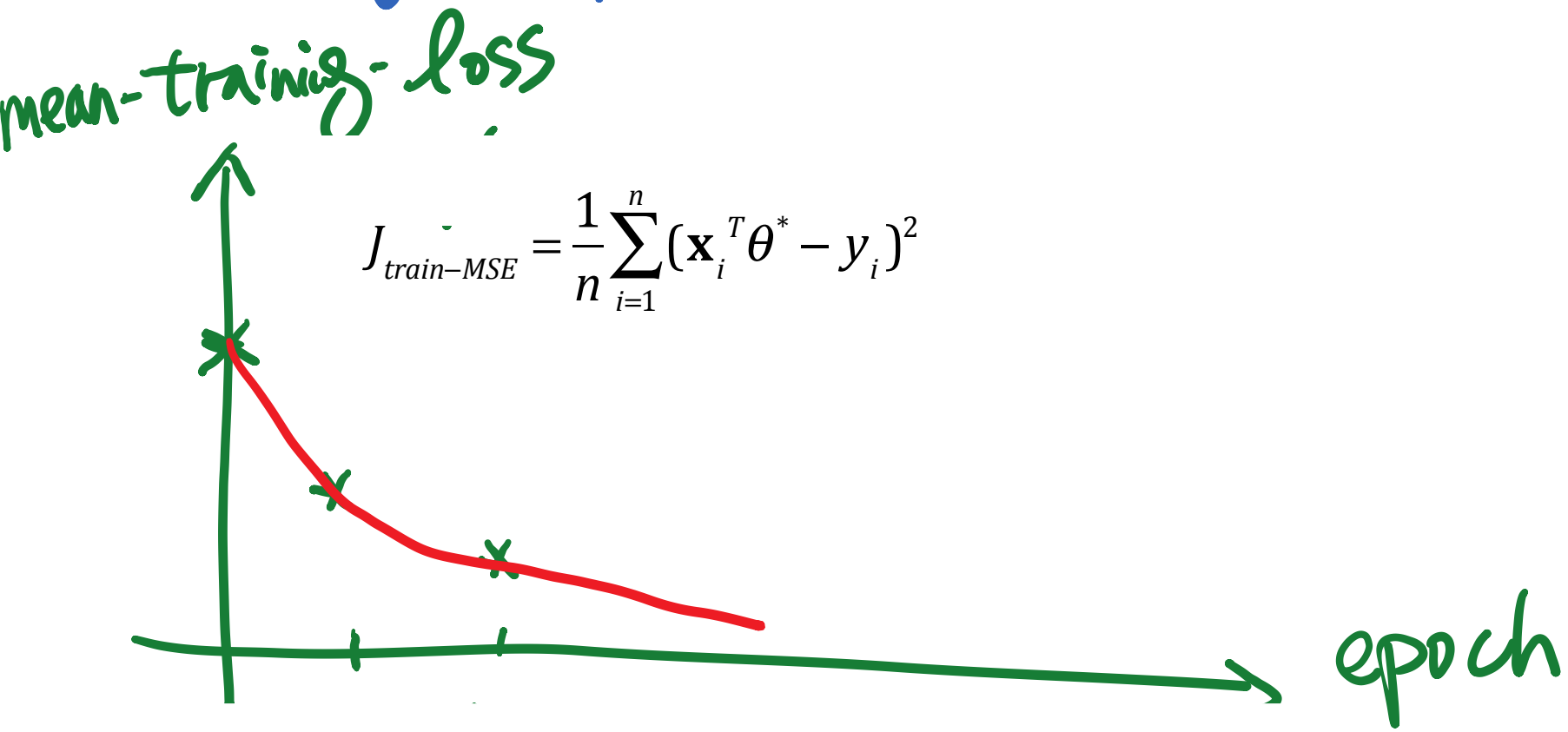
- Train MSE Error to observe:

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

In many situations, visualizing Train-MSE can be helpful to understand the behavior of your method, e.g., the influence of the hyper parameter you chose……

In Homework, when we ask for plots of training error, we ask for the MSE per-sample train errors; Because it is comparable to test MSE error.

One good plot for GD/SGD



Each pass of SGD repeated cycling through all samples in the whole train➔ an epoch !

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

Today Recap & next ➔ Regression (supervised)

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

} variations of $\arg\min_{\theta} L(\theta)$

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

} variations of $f(x)$

→ variations of $L(\theta)$

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

- An overview of gradient descent optimization algorithms, https://arxiv.org/abs/1609.04747

LR with batch GD / Per Feature View

- Note that:

$$\nabla_\theta J = \left[\frac{\partial}{\partial \theta_1} J, \ldots, \frac{\partial}{\partial \theta_k} J \right]^T$$

- For its j-th variable:

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

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

Update Rule Per Feature (Variable-Wise)

LR with Stochastic GD / Per Feature View

- For a single training point (i-th), we have:
  - For its j-th variable:

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

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

Update Rule Per Feature (Variable-Wise)

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 α, or gradually decrease α.

$\alpha_0 = 0.05$

Dr. EricXing's tutorial slide

Extra: 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: Computational Cost (Naïve..)

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

$p \times n$ $n \times p$

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

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

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

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

mostly about Memory Cost →

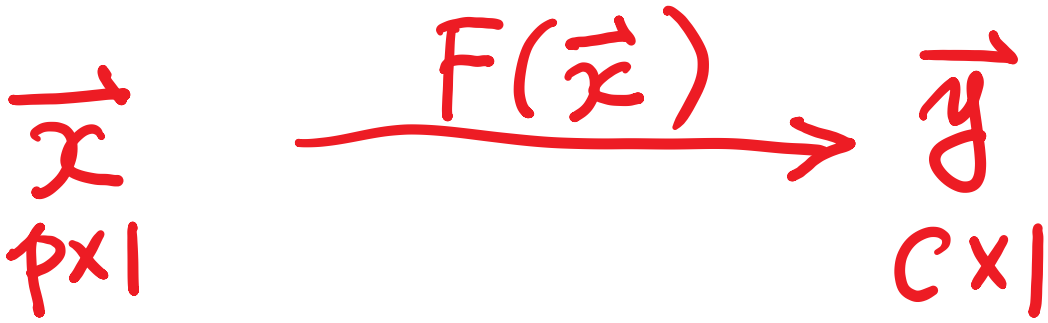
Interesting discussion in: https://stackoverflow.com/questions/10326853/why-does-lm-run-out-of-memory-while-matrix-multiplication-works-fine-for-coeffic

Extra: Newton's Method and

Connecting to Normal Equation

Review: Single Var-Func to Multivariate

Single Var-Function	**Multivariate Calculus**
Derivative	Partial Derivative
	Gradient
Second-order derivative	Directional Partial Derivative
	Vector Field
	Contour map of a function
	Surface map of a function
	Hessian matrix
	Jacobian matrix (vector in / vector out)



$p=1$	$C=1$	derivative	2nd-order derivative
$p>1$	$C=1$	gradient vector	Hessian matrix
$p>1$	$C>1$	Jacobian matrix	

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

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

$p \times 1$

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 !)

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

$$\Downarrow$$

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

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

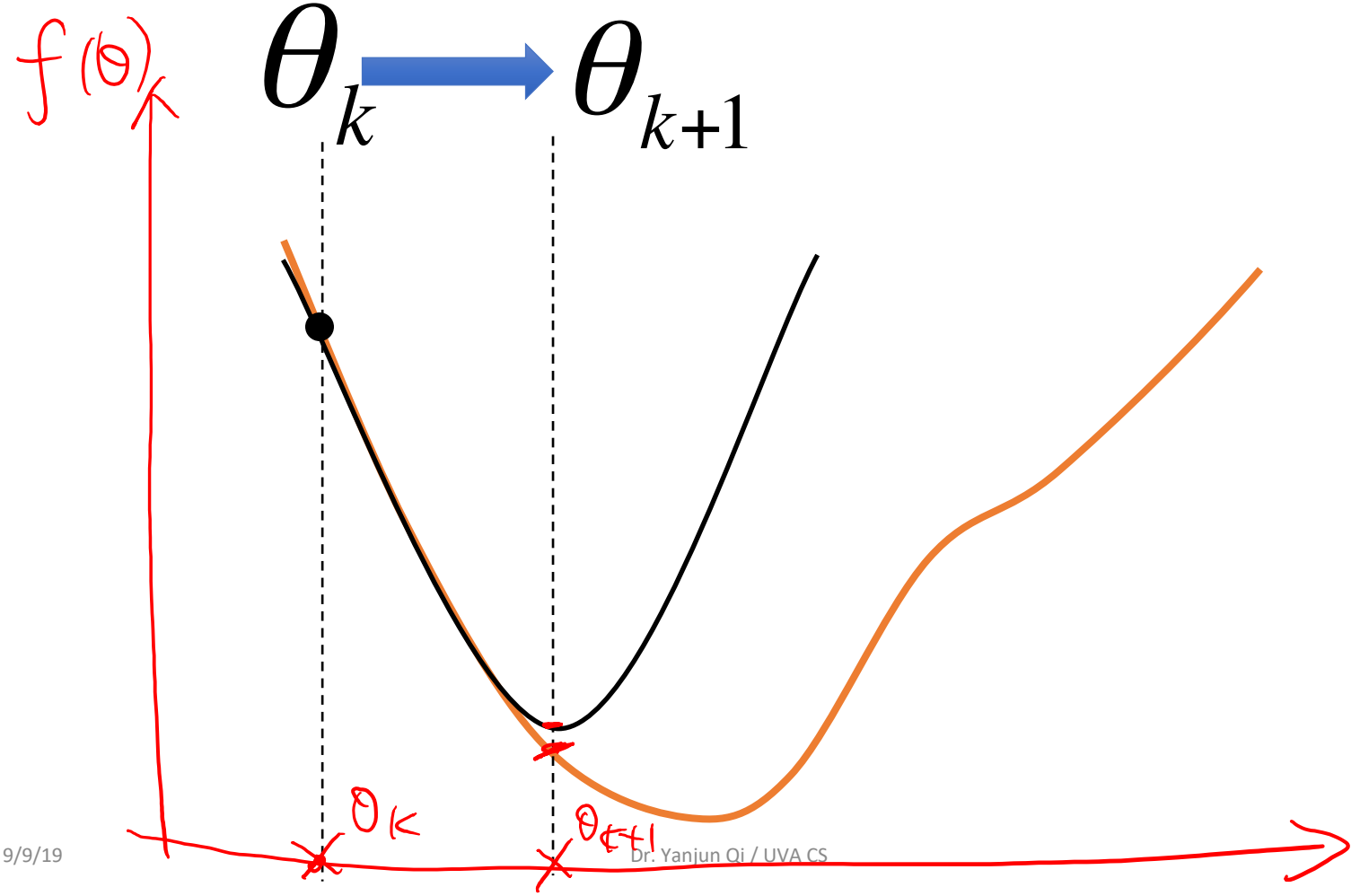
see p24 handout

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

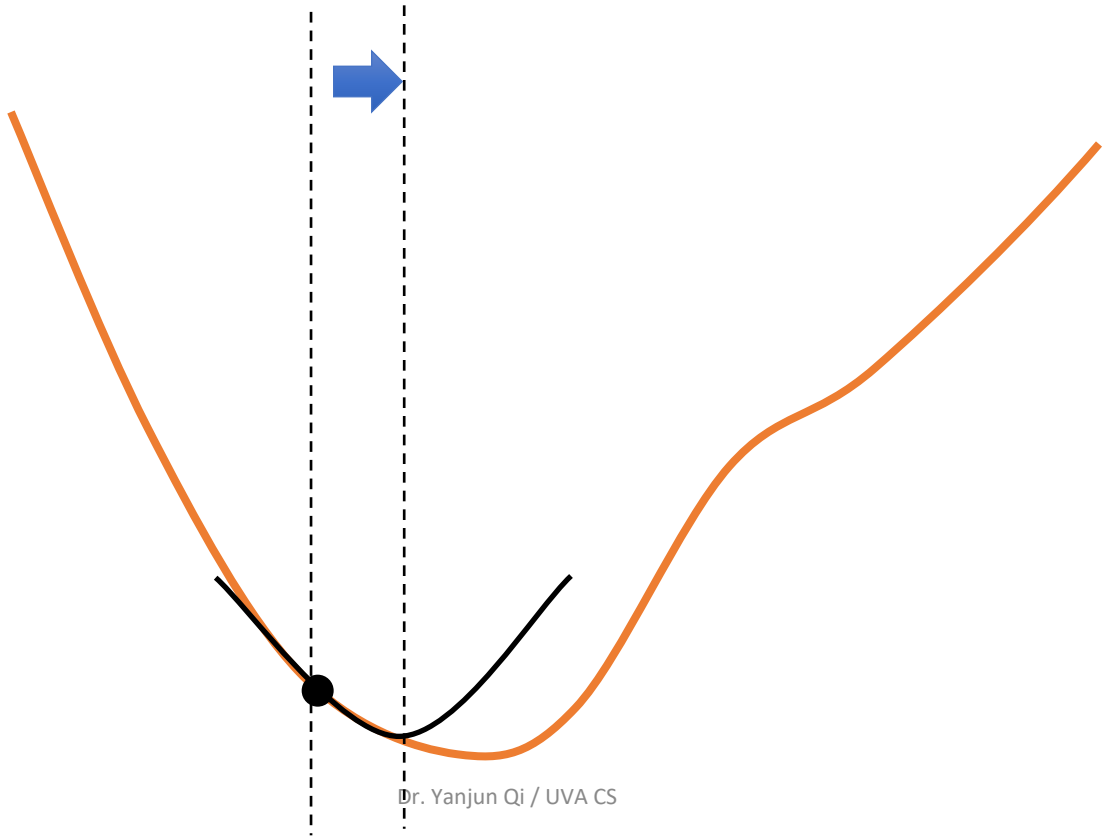
$$\Rightarrow \quad \theta = \theta_k - H_k^{-1} g_k$$

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

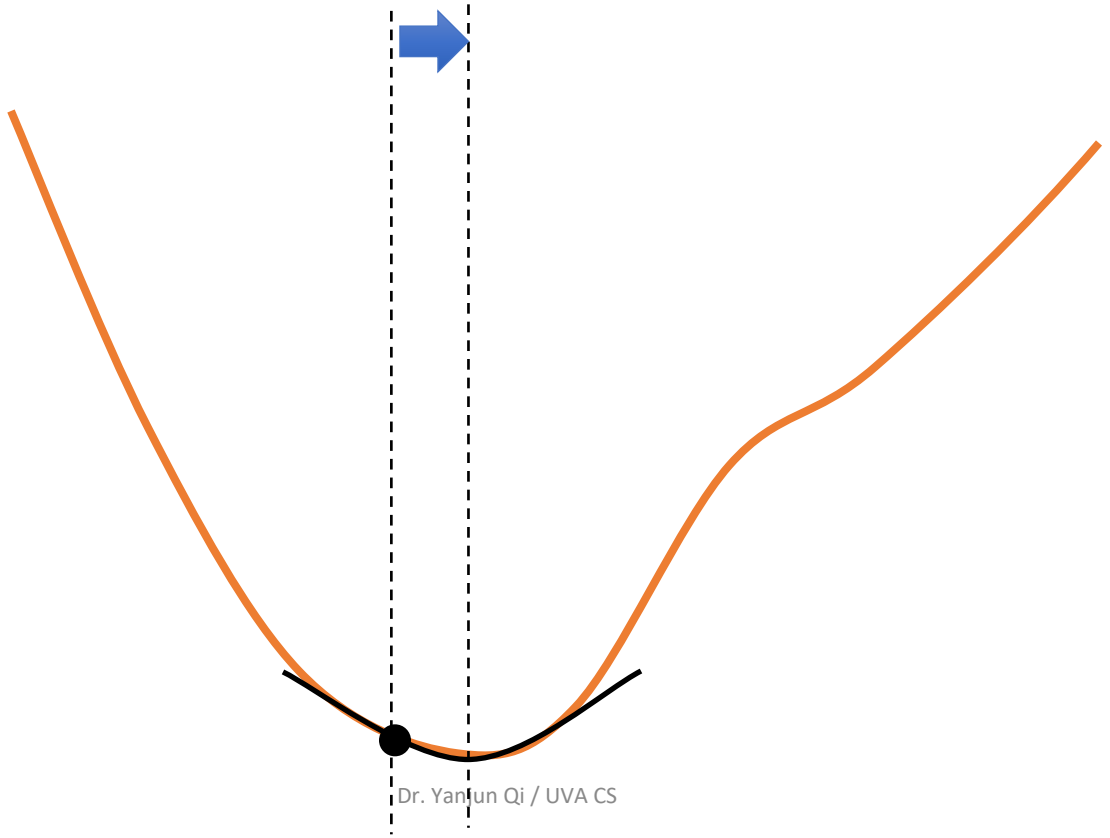
Newton's Method / second-order Taylor series approximation



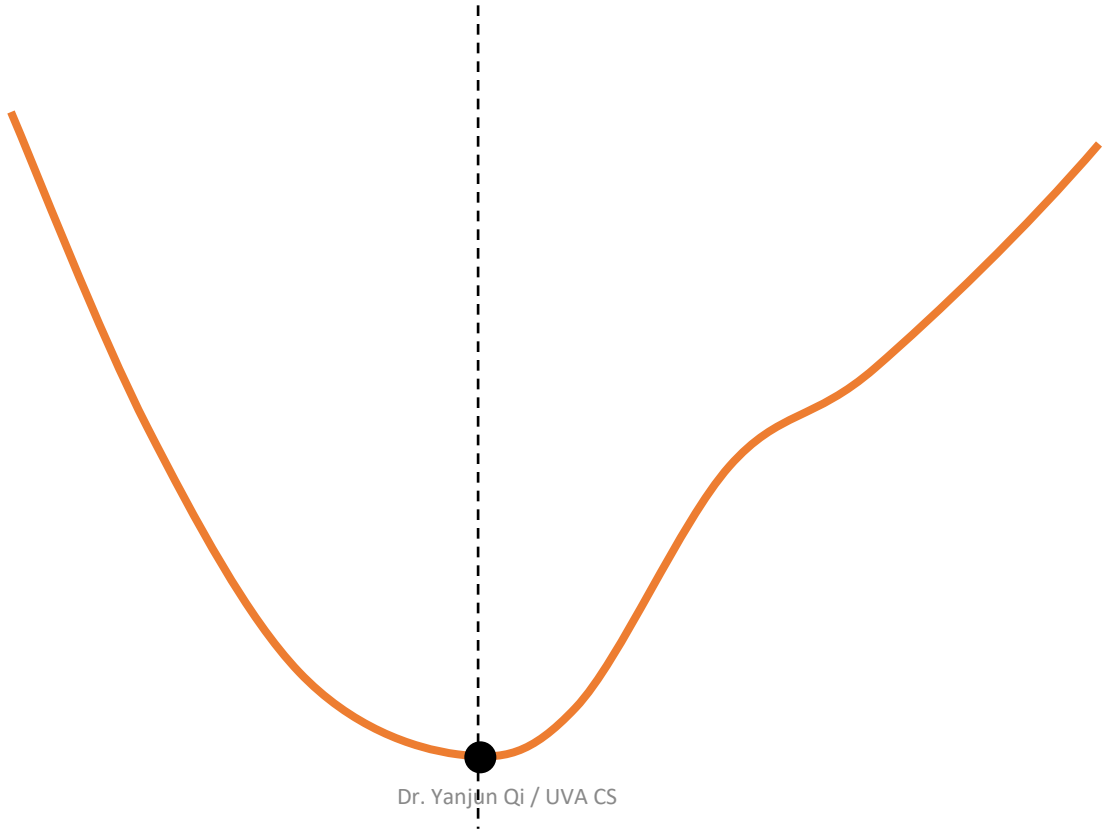
Newton's Method / second-order Taylor series approximation



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

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

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

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

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

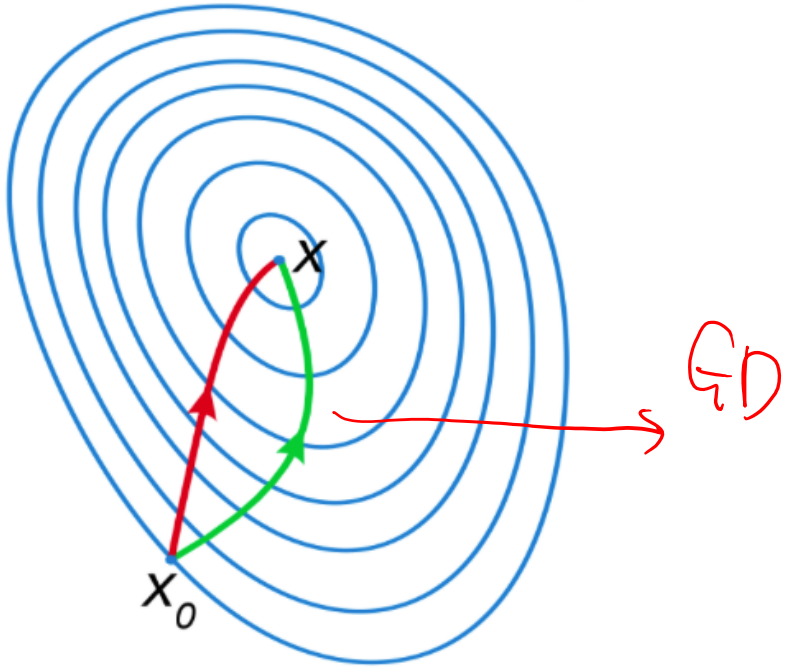
$$\theta_{k+1} = \theta_k - \alpha\, g(\theta_k)$$

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

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

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

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

??? Normal Equation?

Newton's method for Linear Regression