Linear and Polynomial Regression

By Vipul Goyal

This Lecture

Last lecture: we can already implement basic learning algorithms for linear regression with 1 variable. Write code.

This Lecture:

- Gradient Descent for multiple features
- Feature Scaling
- Polynomial regression: Handling higher order dependencies
- Another method for linear/polynomial regression

House Price given Area and Lot size

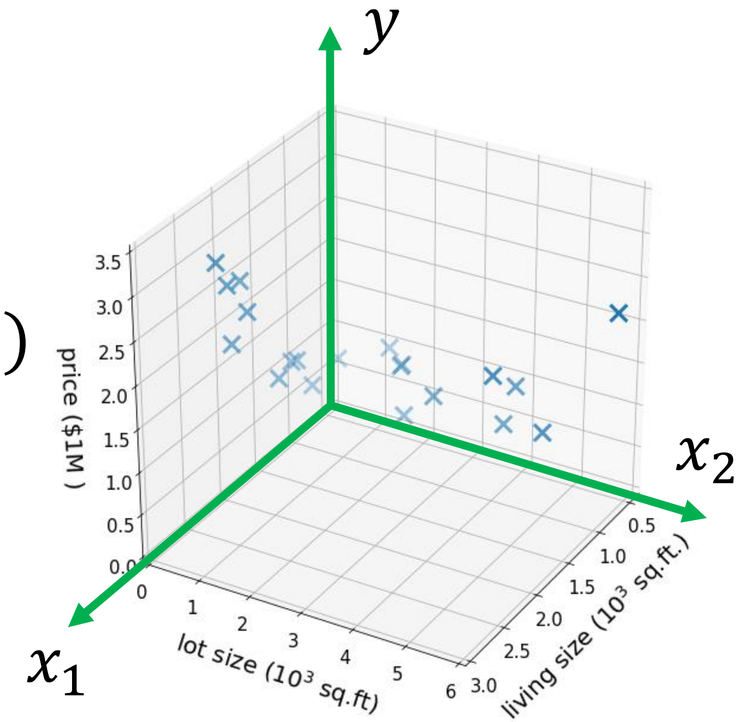
- Task: find a function that maps

$\mathbb{R} = set\ of\ real\ numbers$

$$\underbrace{\text{(size, lot size)}}_{\substack{\text{features/input} \\ x \in \mathbb{R}^2}} \rightarrow \underbrace{\text{price}}_{\substack{\text{label/output} \\ y \in \mathbb{R}}}$$

- Dataset: $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)$

where $x^{(i)} = (x_1^{(i)}, x_2^{(i)})$



High-dimensional (Multiple) Features

- $x \in \mathbb{R}^n$ for large n
- E.g.,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \begin{array}{l} \text{--- living size} \\ \text{--- lot size} \\ \text{--- house age} \\ \text{--- condition} \\ \text{--- zip code} \\ \vdots \\ \\ \end{array} \longrightarrow y \quad \text{--- price}$$

Multiple Features: Some Notation

Multiple features (variables)

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features (n = 4 here)

$x^{(i)}$ = input (features) of i-th training set

$$x^{(1)} = \begin{bmatrix} 2104 \\ 5 \\ 1 \\ 45 \end{bmatrix}$$

$x_j^{(i)}$ = value of feature j in i-th training set

Hypothesis Representation

Previously: $h_\theta(x) = \theta_0 + \theta_1 \cdot x$

Now: $h_\theta(x) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \ldots. + \theta_n \cdot x_n$

where $$x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ . \\ x_n \end{bmatrix}$$

If we need to refer to a specific (say i-th) training set, we write $x^{(i)}$. Else simply x.

Notational Change

$$h_\theta(x) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + ... + \theta_n \cdot x_n$$

To make notation simpler, define x_0 = 1. Equation becomes:

$$h_\theta(x) = \theta_0 . x_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + ... + \theta_n \cdot x_n$$

Writing as per Matrices:

$$\text{x} = \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ . \\ x_n \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ . \\ . \\ . \\ . \\ \theta_n \end{bmatrix}$$

Now: $h_\theta(x) = \theta^{\mathsf{T}} . \text{x}$

Matrix Notation

Compute: $h_\theta(x) = \theta^{\mathsf{T}} \cdot x$

$$= [\theta_0\ \theta_1\ \dots\ \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ . \\ x_n \end{bmatrix}$$

$$= \theta_0 . x_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \dots + \theta_n \cdot x_n$$

Defining the Cost Function

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 \dots + \theta_n x_n$

Parameters: $\theta_0, \theta_1 \dots \theta_n$

Cost function: $J(\theta_0, \theta_1 \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

$\theta_0, \theta_1, \dots, \theta_n$ will simply be denoted by θ

How do we minimize this cost function?

Gradient Descend for Multiple Variables

Gradient Descent for Multiple Variables

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 \; .... + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots \theta_n$

Cost function:

$$J\,(\theta_0, \theta_1 .... \,\theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta\left(x^{(i)}\right) - y^{(i)})^2$$

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

GD for One Variables: Notation Change

$\cancel{h_\theta(x) = \theta_0 + \theta_1 \cdot x}$

$h_\theta(x) = \theta_0 . x_0 + \theta_1 \cdot x_1$

with $x_0 = 1$

Gradient descent algorithm

repeat until convergence

$$\left\{ \theta_j := \theta_j - \alpha \boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)} \right\}$$

(for j = 1 and j = 0)

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\frac{d}{d\theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

General Partial Derivative Equation

$$\frac{d}{d\theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_0^{(i)}$$

$$\cdots$$

$$\frac{d}{d\theta_n} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_n^{(i)}$$

Gradient descent:

Repeat $\left\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n) \right\}$

(simultaneously update for every $j = 0, \ldots, n$)

Gradient Descent for Multiple Variables

New algorithm (n ≥ 1):

Repeat $\{\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$

(simultaneously update θ_j for j = 0, …, n)

$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_2^{(i)}$$

...

Summary

- Linear Regression to multiple variables can be solved by generalizing gradient descent

- More realistic to have multiple variables rather than 1. Typically, what you are trying to predict is complex and depends on many factors.

- It have wide number of applications (stocks,…)

Polynomial Regression

Choice of Features Matter!

Housing prices prediction?

$$h_\theta(x) = \theta_0 + \theta_1 \cdot frontage + \theta_2 \cdot depth$$

Frontage (feet) X_1	Depth (feet) X_2	Price (\$1000) y
30	15	460
24	18	500
25	19	515
28	20	578

We will never get the right prediction with linear regression!

Idea: Add an Extra Feature

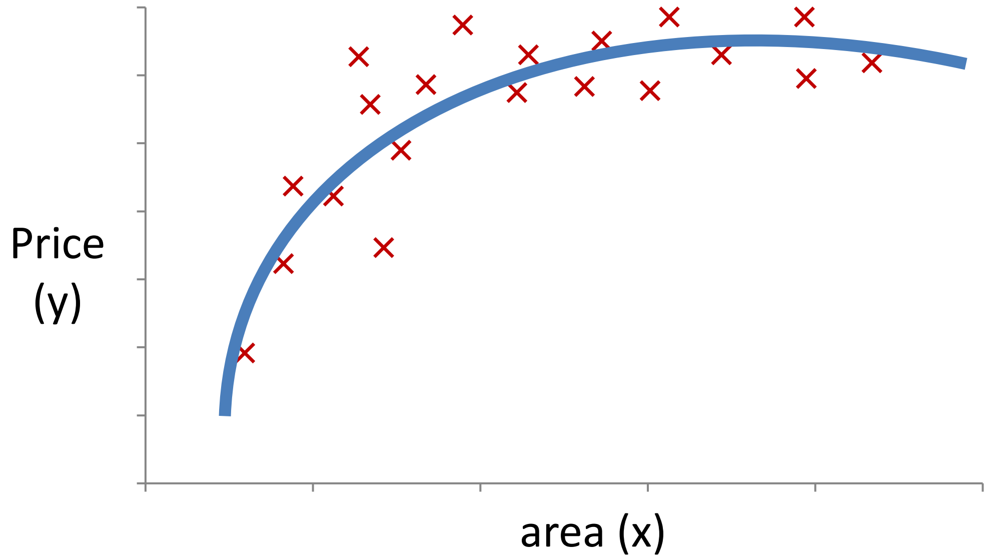
Add another column to the table

$$h_\theta(x) = \theta_0 + \theta_1 \cdot frontage + \theta_2 \cdot depth$$

$$+\theta_3(area)$$

Frontage X_1	Depth X_2	Area $X_1 \cdot X_2$	Price Y
30	15	450	460
24	18	432	232
25	19	475	315
28	20	560	178
…	…	…	…

Polynomial Regression



$$h_\theta(x) = \theta_0 + \theta_1(area) - \theta_2(age)$$

$$h_\theta(x) = \theta_0 + \theta_1(area) - \theta_2\sqrt{age}$$

(Add another column for $\sqrt{age}$)

Feature Scaling

Features can have Different Ranges

$$h_\theta(x) = \theta_0 . x_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + ... + \theta_n \cdot x_n$$

$$x = \begin{bmatrix} 3000 \\ 4 \\ . \\ . \\ . \end{bmatrix} \begin{array}{l} \text{--- area (0-4000 sq feet)} \\ \text{--- number of bedrooms (1-5)} \\ \vdots \\ \\ \\ \end{array}$$

Typically: change of 1 in bedroom more "significant" to price than change of 1 in area

Car Prices

$$h_\theta(x) = \theta_0 . x_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + ... + \theta_n \cdot x_n$$

$$x = \begin{bmatrix} 70000 \\ 4 \\ . \\ . \\ . \end{bmatrix} \begin{array}{l} \text{--- Miles Driven (0-100K)} \\ \text{--- Age of car (0 – 20 years)} \\ \vdots \\ \\ \\ \end{array}$$

Again: change of 1 in age more "significant" to price than change of 1 in miles

Hence: θ_2 needs to be much larger than θ_1

More work for Gradient Descent: need to search a variety of ranges for each θ_i. Can we do better?

Feature Scaling

Feature Scaling

Get every feature into approximately a $-1 \le x_i \le 1$ range.

$x_0 = 1$

$0 \le x_1 \le 3$

$-2 \le x_2 \le 0.5$

$-100 \le x_3\ 100$

$-0.001 \le x_4 \le 0.001$

How to Scale?

Step 1: Replace x_i with x_i - μ_i to make sure features have approximately zero mean (exclude x_0)

x_1 = size – 1500 (if average size = 1500)

x_1 = age – 5 (if average age of car = 5)

Step 2: Divide by an appropriate number to make sure range is roughly from -1 to 1 (roughly max – min)

$$x_1 = \frac{\text{size} - 1500}{2000}$$ (if max size = 3500)

$$x_1 = \frac{\text{age} - 5}{10}$$ (if max age of car = 15)

Note: doesn’t have to be exact, remember its only an optimization

Feature Scaling Observations

- Why -1 to 1 range for x_i? Choice somewhat arbitrary.
- Based on the assumption that if ranges for x_i are similar, ranges for optimal θ_i will also be similar.
- In general: quickest convergence if the optimal values for θ_i all in a similar range (because gradient descent will start with similar values for all θ_i)
- Don't need to search widely different values for different θ_i

Learning Rate

Running Gradient Descent Correctly

Gradient descent

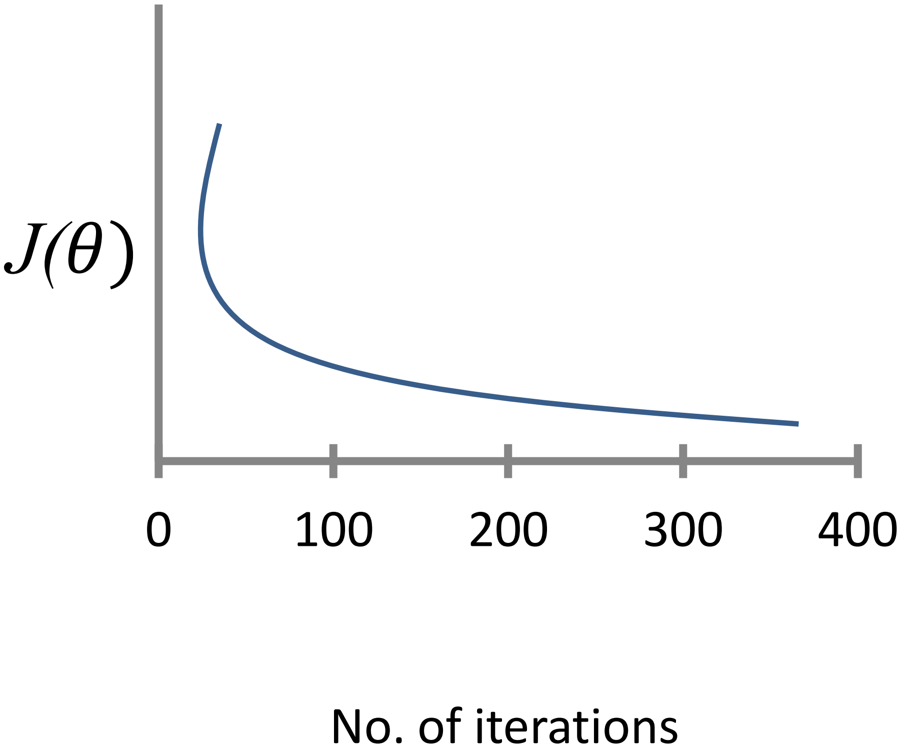
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- "Debugging": How to make sure gradient descent is working correctly.

- How to choose learning rate α.

A "Good" Execution

Cost *J* must keep going down with number of iterations



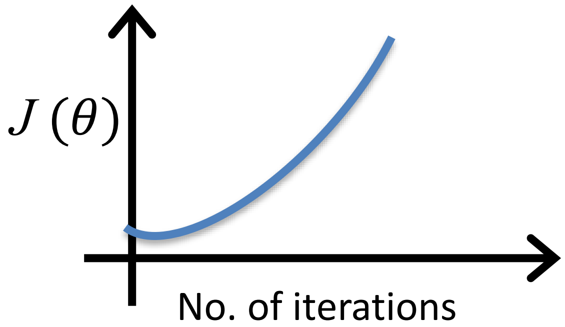
Example automatic convergence test:

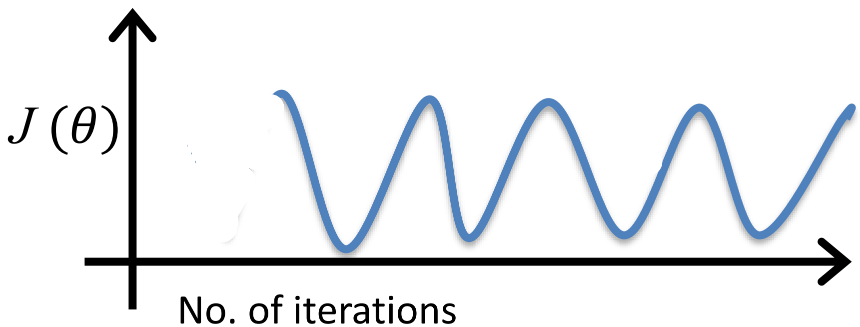
Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

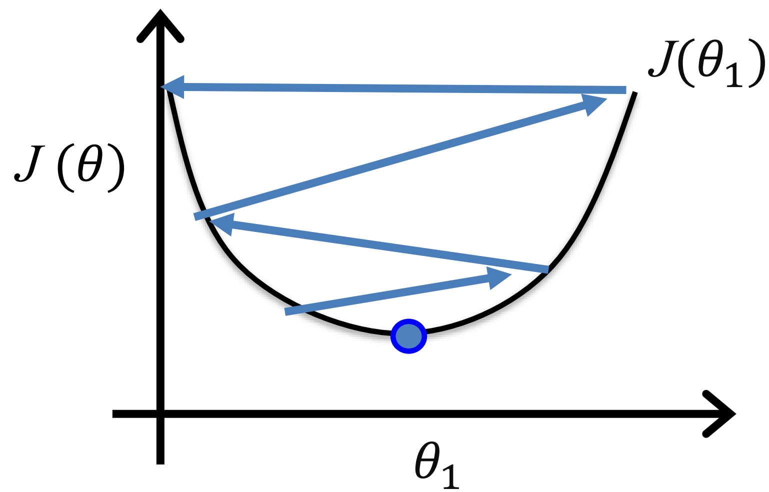
A “Bad” Execution

Gradient descent not working.

Use smaller α.







- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Choosing the Learning Rate

Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α, try

$$\ldots, 0.001, \qquad , 0.01, \qquad , 0.1, \qquad , 1, \ldots$$

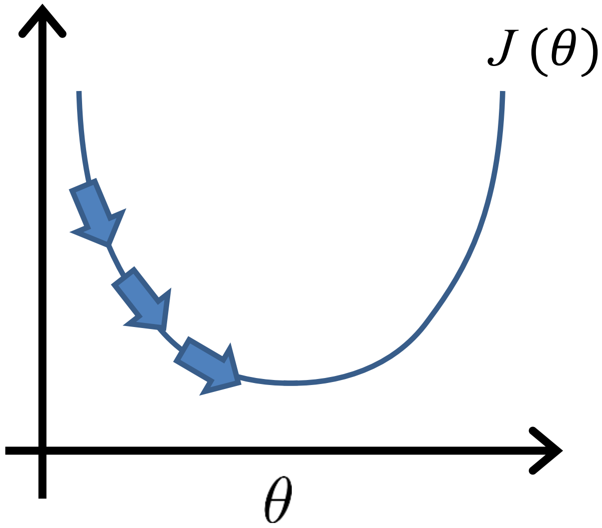
Another Idea

- Store the current cost function value $J(\theta)$
- Run the next step of gradient descent
- Compute the new cost function value $J(\theta)$ with the new values of θ
- If cost function went down, proceed as usual
- Otherwise, go back to older values of θ, reduce learning rate and try again
  - Good heuristic: divide α by 2 or 10

Linear Regression without Gradient Descent

Solving Linear Regression

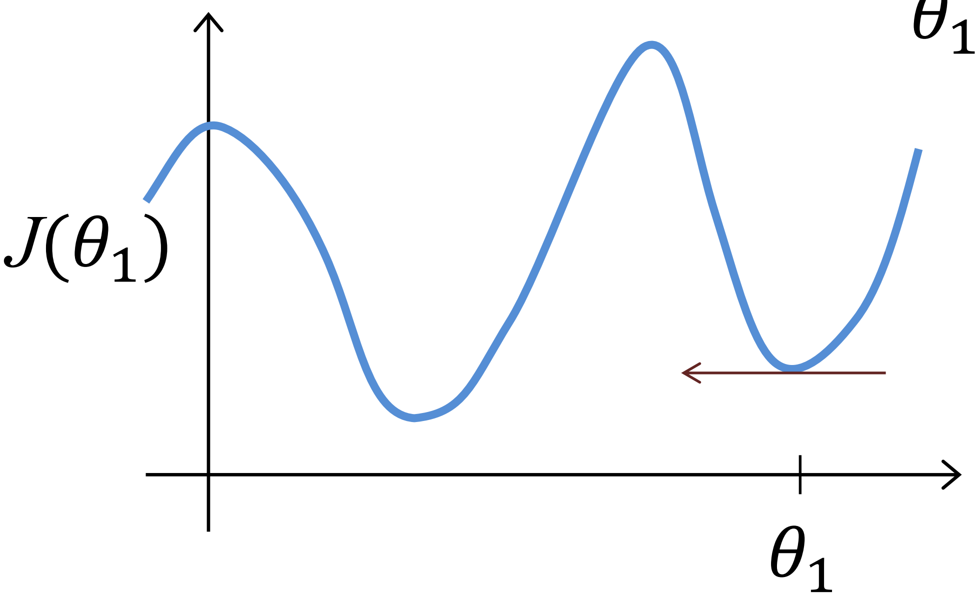
Gradient Descent



Normal equation: Method to solve for θ directly

Key Idea of Normal Equation Method

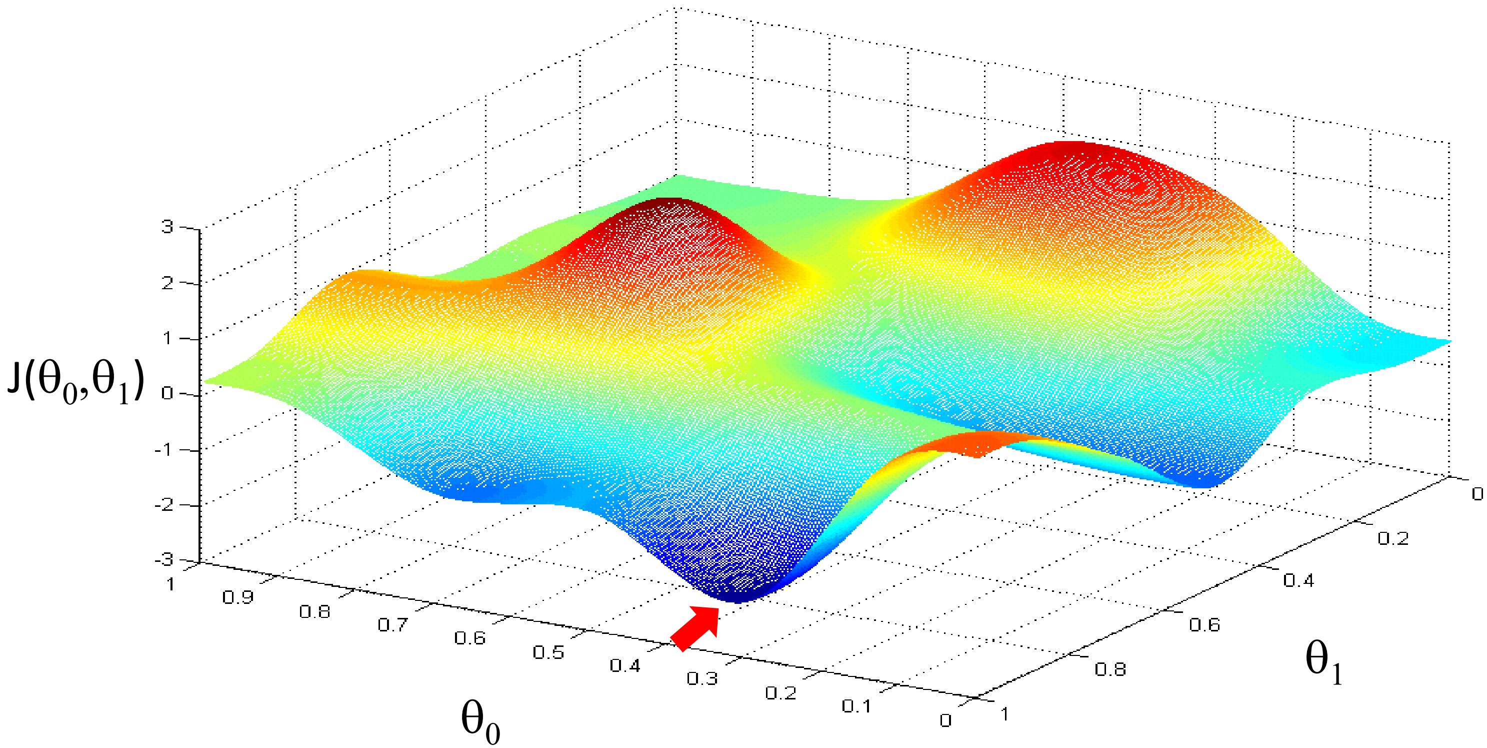
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$



Derivative is 0 at local minima and maxima. Why?

- The function changes direction: first it was reducing, then increasing (or vice versa)

Multidimensional Cost Function



ALL Derivative 0 at lowest (and highest) points

Normal Equation Method

Mathematically compute points where all derivatives are 0

$$\frac{d}{d\theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_0^{(i)} = 0$$

$$\frac{d}{d\theta_1} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_1^{(i)} = 0$$

$$\ldots$$

$$\frac{d}{d\theta_n} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_n^{(i)} = 0$$

So we have n+1 variables $\theta_0, \theta_1 \ldots \theta_n$ (x, y known) and n+1 equations!

Simplifying Further

Remember: $h_\theta(x) = \theta^\top \cdot \mathrm{x}$

$$\frac{1}{m}\sum_{i=1}^{m}(\theta^\top \cdot x^{(i)} - y^{(i)})\, x_0^{(i)} = 0$$

…

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

X = training example matrix (with m rows and n+1 columns)

$\left(X^T X\right)^{-1}$ is inverse of matrix $X^T X$

An Example

Examples: $m = 5.$

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
1	3000	4	1	38	540

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ 1 & 3000 & 4 & 1 & 38 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ 540 \end{bmatrix}$$

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

Computing The Solution(s)

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

Sanity check: *X* is *mxn'* *where n' = n+1*

*n'*x1 = (*n'*x*m*) (*m*x*n'*) (*n'*x*m*) (*m*x1)

*n'*x1 = (*n'*x*n'*) (*n'*x1)

*n'*x1 = *n'*x1

Keep in mind:

- Computing matrix multiplication matrix inverse can be expensive

A Comparison

m training examples, n features.

<u>Gradient Descent</u>

- Need to choose α
- Needs many iterations
- Works well even when n is large
- May not give the absolute best result

<u>Normal Equation</u>

- No need to choose α
- Don't need to iterate
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large
- Might give best possible result

If working with big datasets, gradient descent might be a better choice

Questions?