



Linear regression with gradient descent

Jay Urbain, PhD

Credits: Nando de Freitas, Oxford; Andrew Ng, Stanford; Hastie and Tibshirani, Stanford

Linear Regression with Gradient Descent

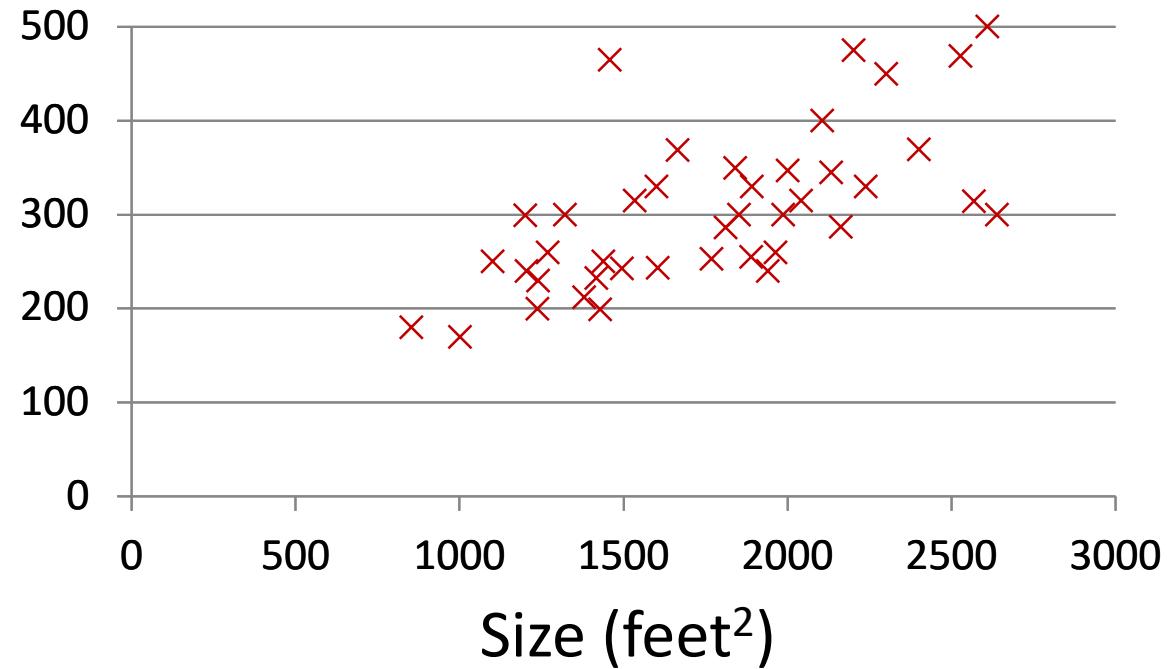
- Linear Regression
 - Hypothesis formulation, hypothesis space
- Optimizing Cost with Gradient Descent
- Using multiple input features with Linear Regression

Notes on:

- Feature Scaling
- Nonlinear Regression
- Optimizing Cost using derivatives

Housing Prices (Milwaukee, WI)

Price
(in 1000's
of dollars)



Supervised Learning

Given the “right answer” for each example in the data.

Regression Problem

Predict real-valued output

Training set of housing prices (Milwaukee, WI)

Size in feet ² (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

Notation:

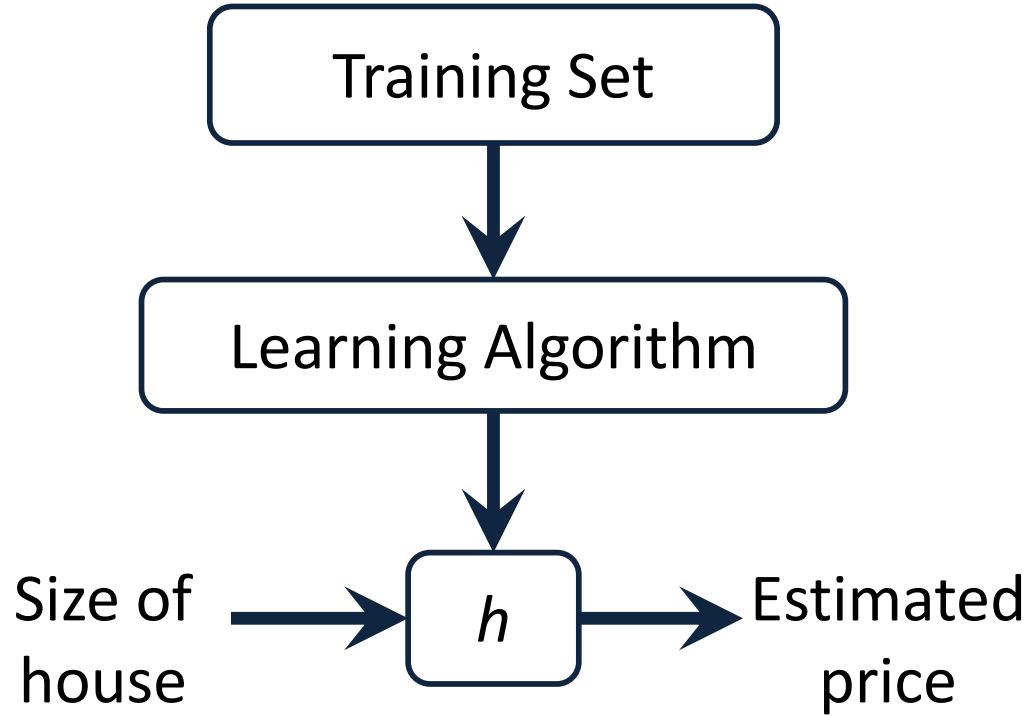
m = Number of training examples $x^{(1)} = 2104$

x's = “input” variable / features $x^{(2)} = 1416$

y's = “output” variable / “target” variable \dots

(**x**, **y**) – one training example $y^{(1)} = 460$

($x^{(1)}, y^{(1)}$) – ith training example



*h – hypothesis to learn (parameterized model)
h maps from x to y*

How do we represent h ?

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

Linear regression with one variable.
(Univariate linear regression)

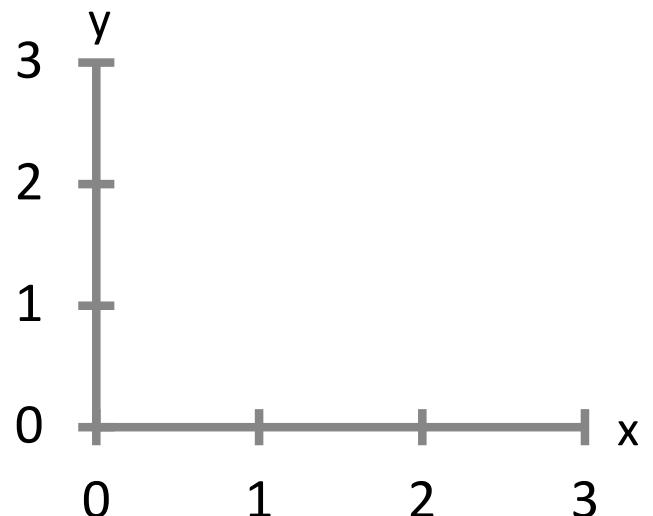
Training Set	Size in feet ² (x)	Price (\$) in 1000's (y)
	2104	460
	1416	232
	1534	315
	852	178

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

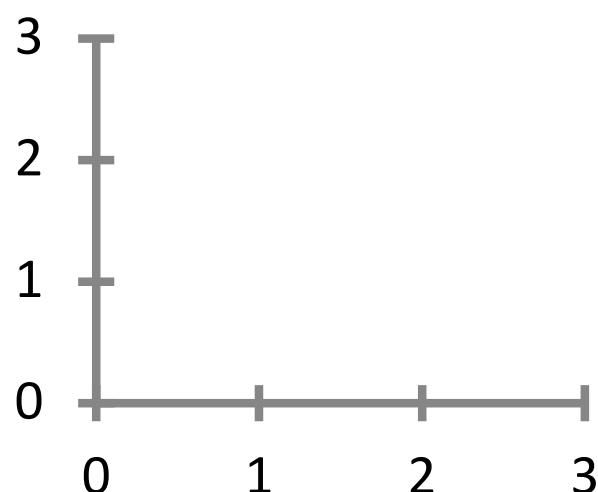
θ_i 's: Parameters

How to choose θ_i 's ?

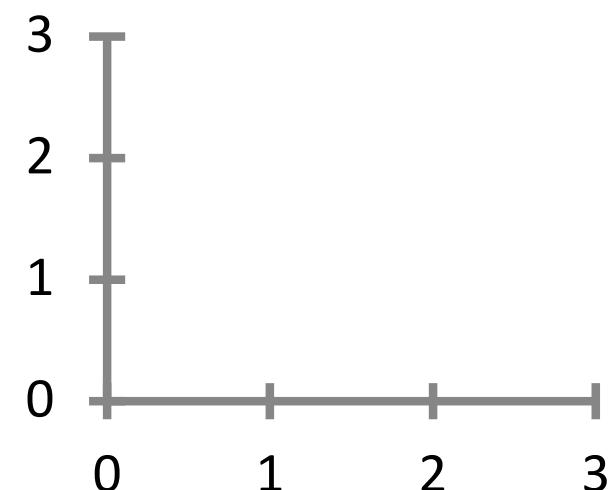
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



$$\begin{aligned}\theta_0 &= 1.5 \\ \theta_1 &= 0\end{aligned}$$



$$\begin{aligned}\theta_0 &= 0 \\ \theta_1 &= 0.5\end{aligned}$$



$$\begin{aligned}\theta_0 &= 1 \\ \theta_1 &= 0.5\end{aligned}$$

Draw $h(x)$

Simplified

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

Goal: minimize $J(\theta_0, \theta_1)$

$$h_{\theta}(x) = \theta_1 x$$

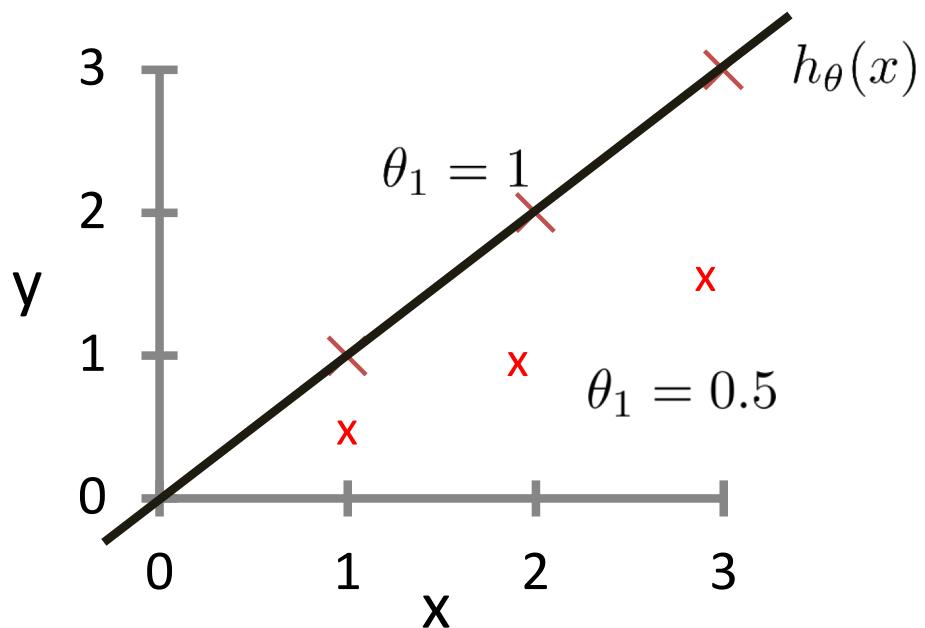
$$\theta_1$$

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

minimize $J(\theta_1)$

$$h_{\theta}(x)$$

(for fixed θ_1 , this is a function of x)

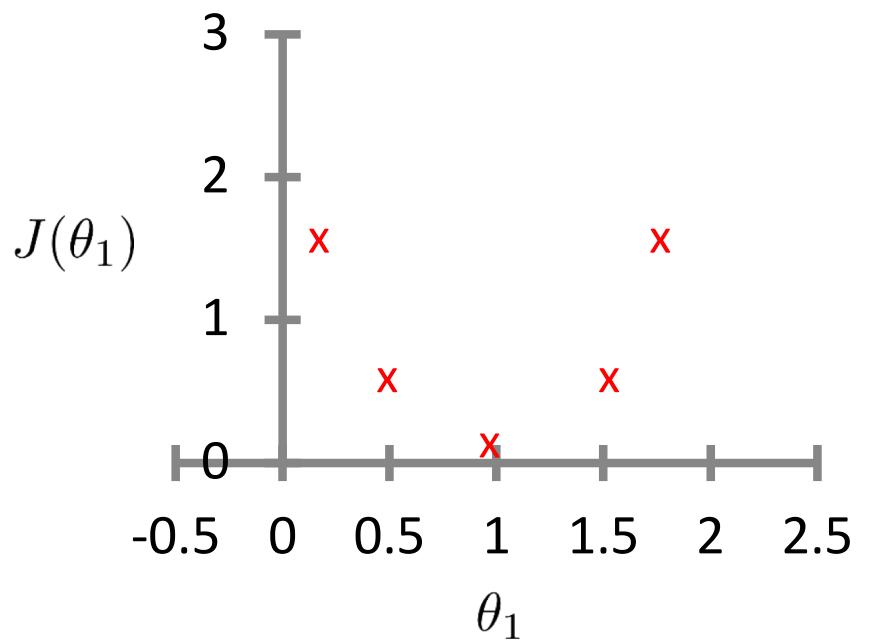


$$h_{\theta}(x) = \theta_1 x$$

Note: $h(x^{(i)}) - y^{(i)} = 0$, $J^{(i)} = 1/(2m) * (0^2 + 0^2 + 0^2) = 0$

$$J(\theta_1)$$

(function of the parameter θ_1)



$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

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

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$

Fitting data well: least squares cost function

In regression, you almost always want to fit the data well

- smallest average distance to points in training data
($h(x)$ close to y for (x, y) in training data)

Cost function J :

$$\begin{aligned} J(\theta_0, \theta_1) &= \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{2m} \sum_{i=1}^m ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2 \end{aligned}$$

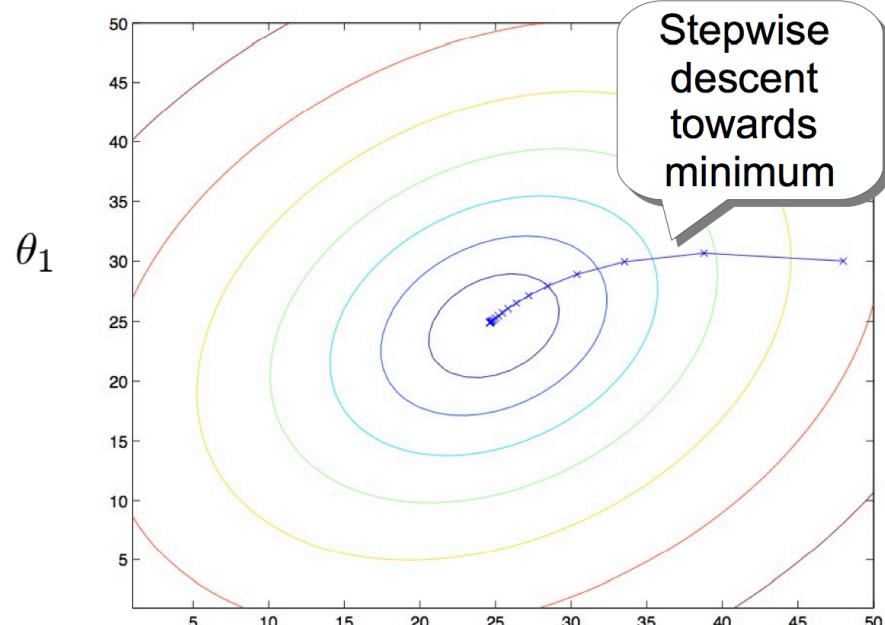
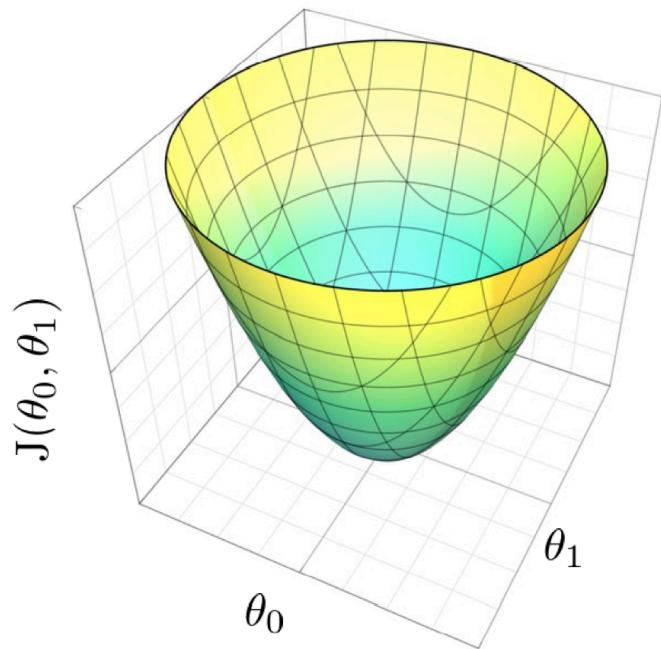
Number of
training instances

Squaring

- Penalty for positive and negative deviations the same
- Penalty for large deviations stronger

Optimizing Cost with Gradient Descent

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m ((\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$$



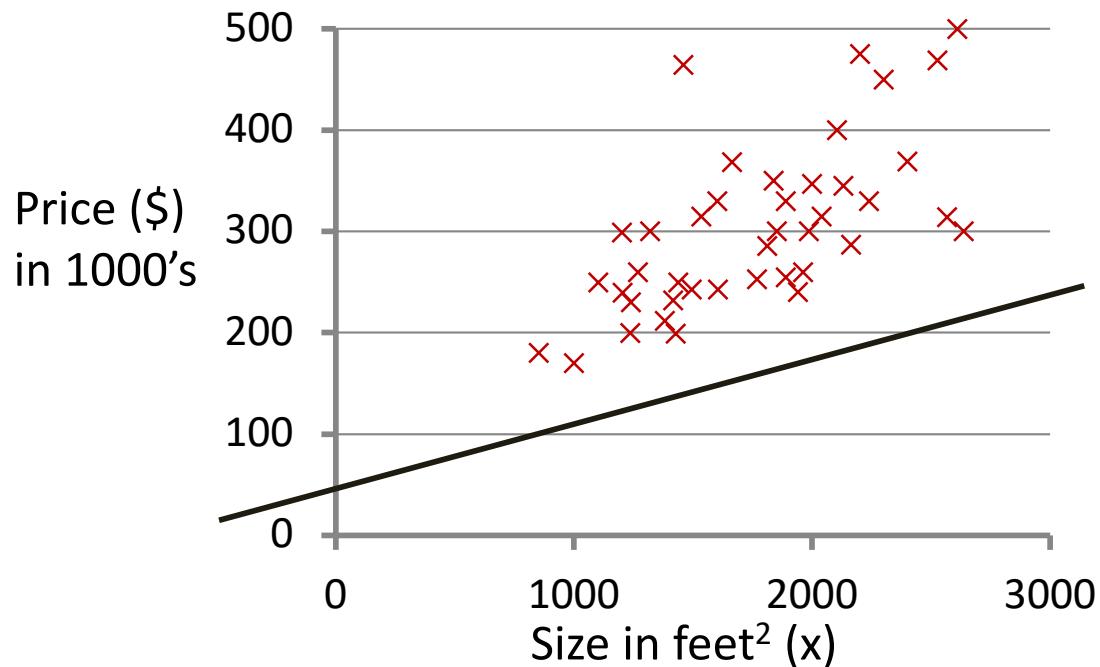
Derivatives
work only for
few parameters

$\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1) = \arg \underset{\theta}{\min} J(\theta)$

Stepwise
descent
towards
minimum

$$h_{\theta}(x)$$

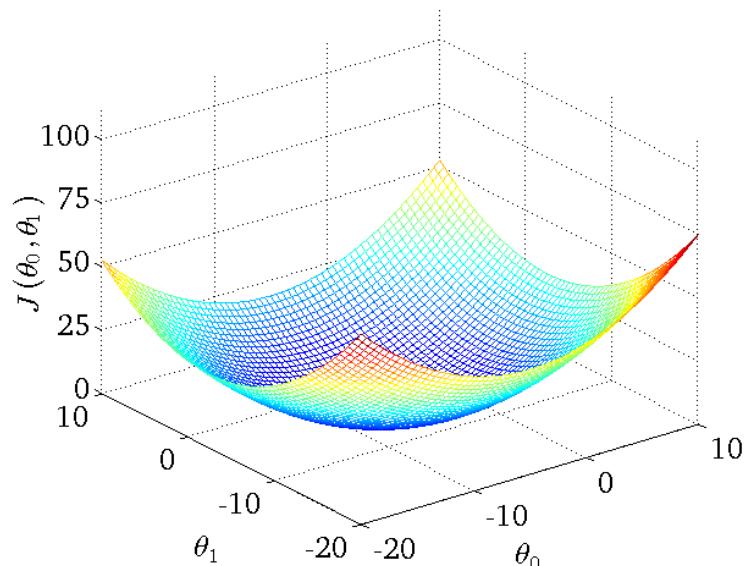
(for fixed θ_0, θ_1 , this is a function of x)



$$h_{\theta}(x) = 50 + 0.06x$$

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

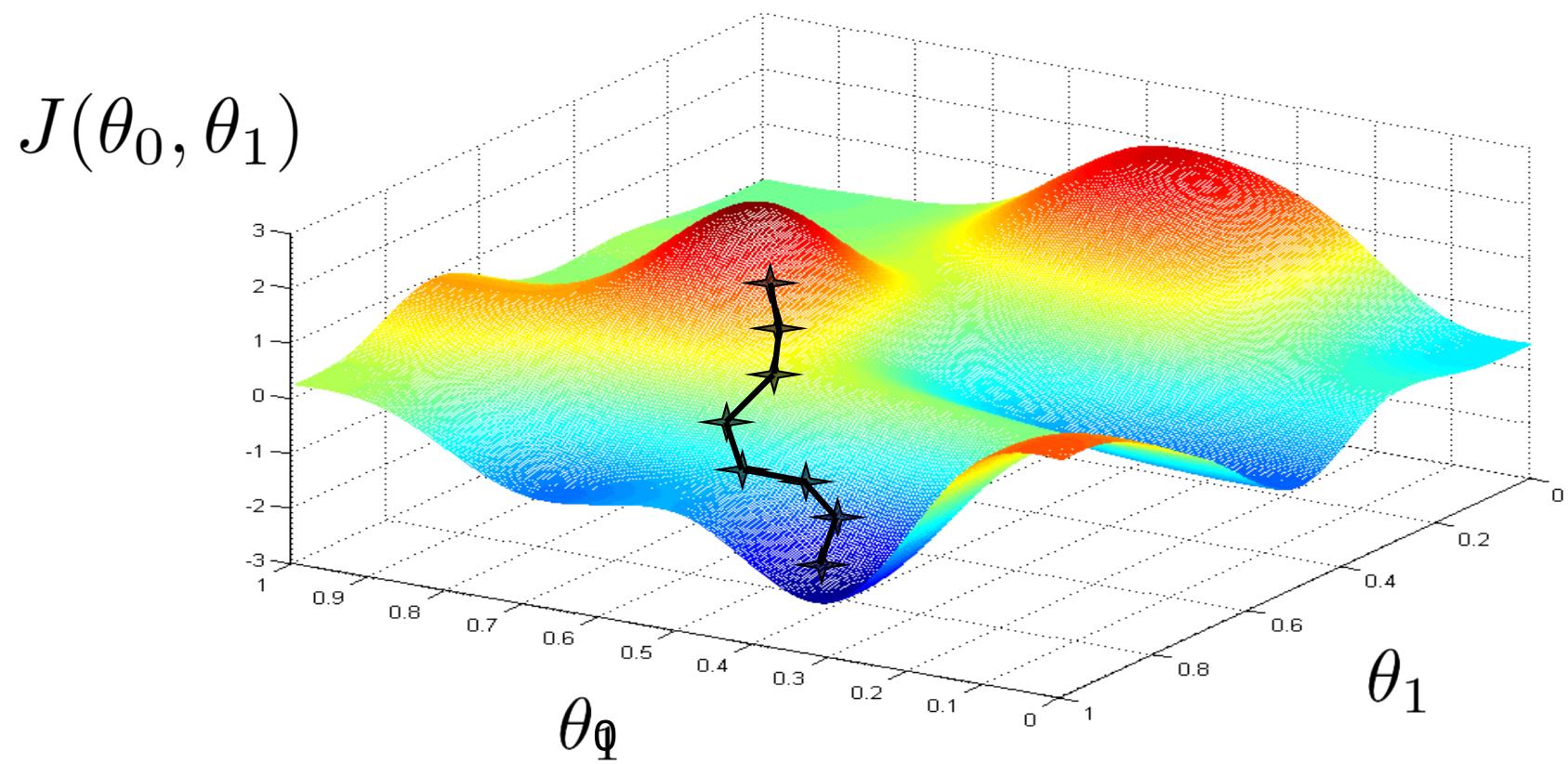


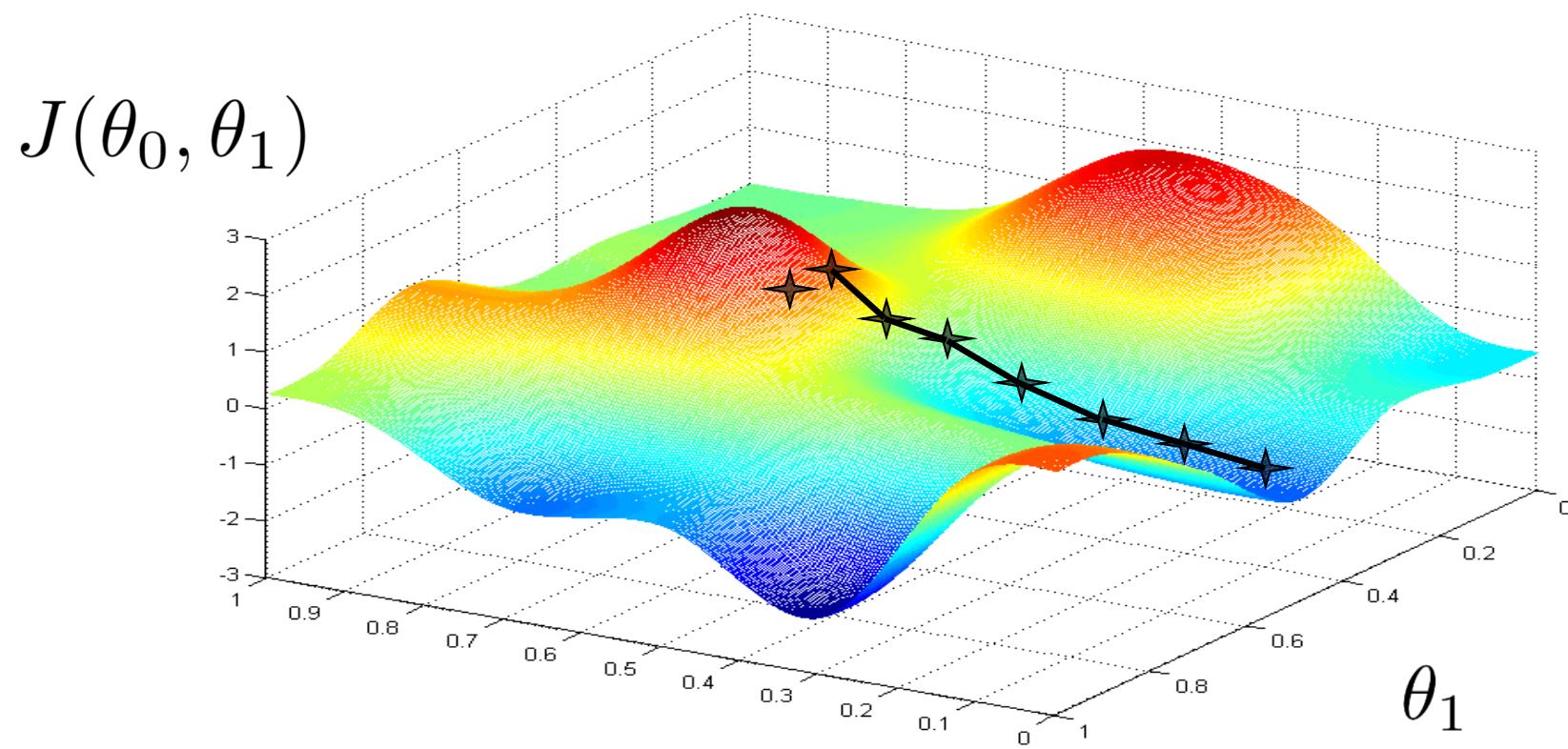
Have some function $J(\theta_0, \theta_1)$

Want to $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
 - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
- until we hopefully end up at a minimum *How?*





Gradient descent algorithm

```
repeat until convergence {  
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$     (for  $j = 0$  and  $j = 1$ )  
}
```

Correct: Simultaneous update

```
temp0 :=  $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$   
temp1 :=  $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$   
 $\theta_0 := \text{temp0}$   
 $\theta_1 := \text{temp1}$ 
```

Incorrect:

```
temp0 :=  $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$   
 $\theta_0 := \text{temp0}$   
temp1 :=  $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$   
 $\theta_1 := \text{temp1}$ 
```

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{simultaneously update } j = 0 \text{ and } j = 1)$$

}

learning rate

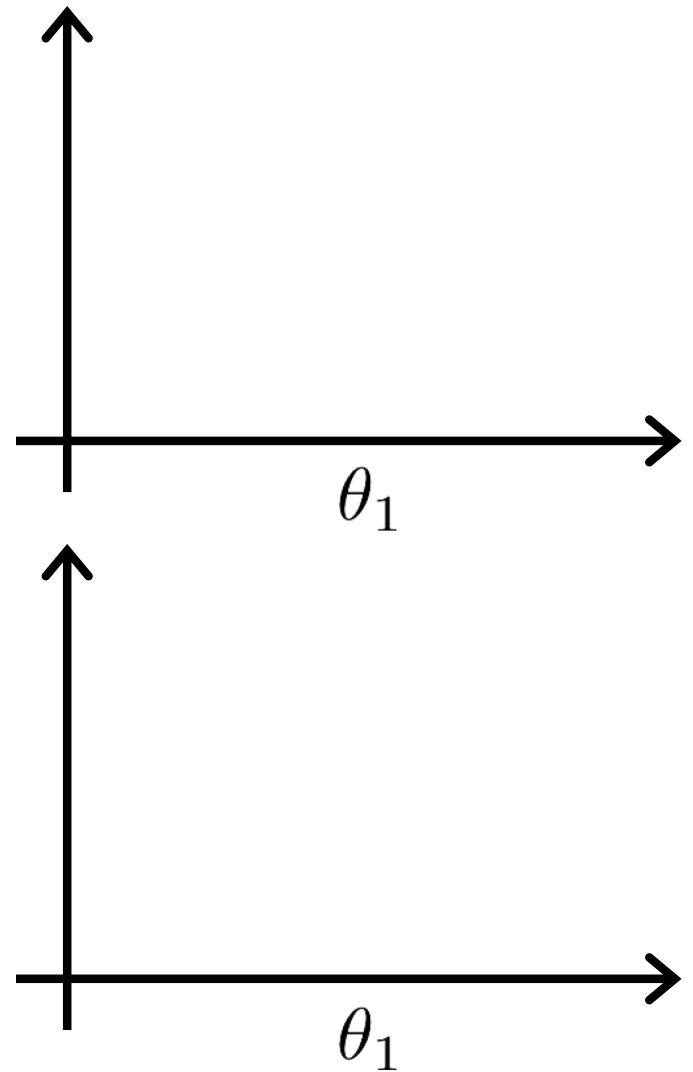
derivative

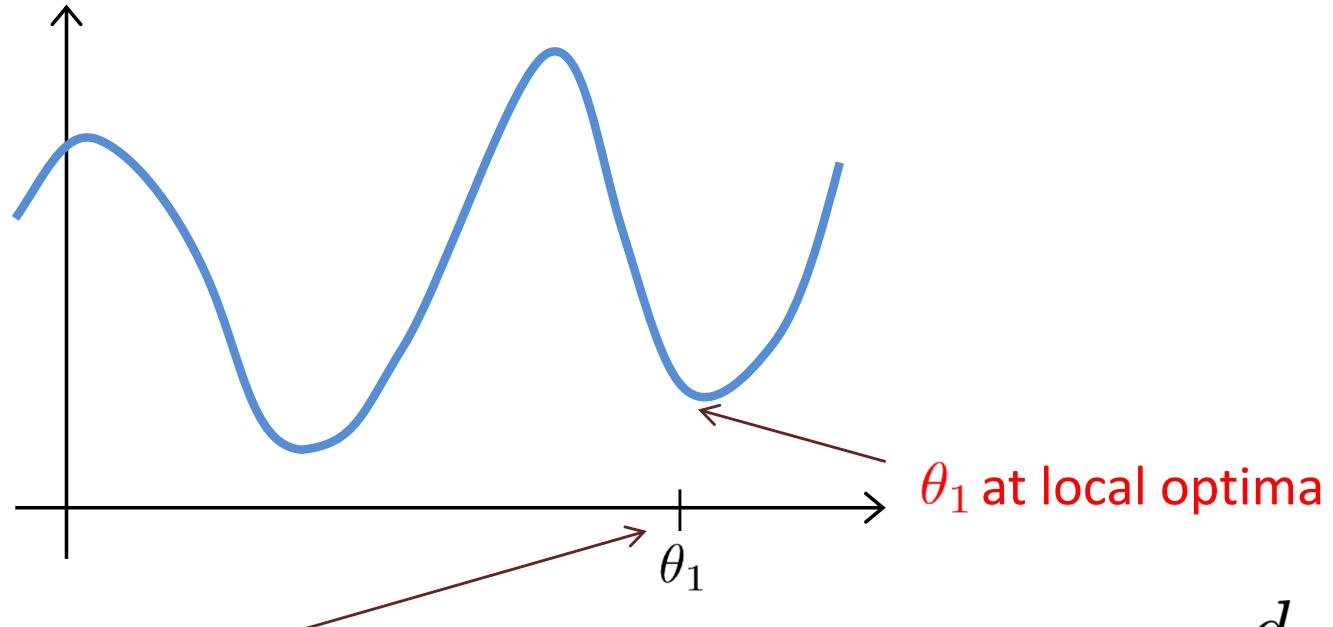


$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



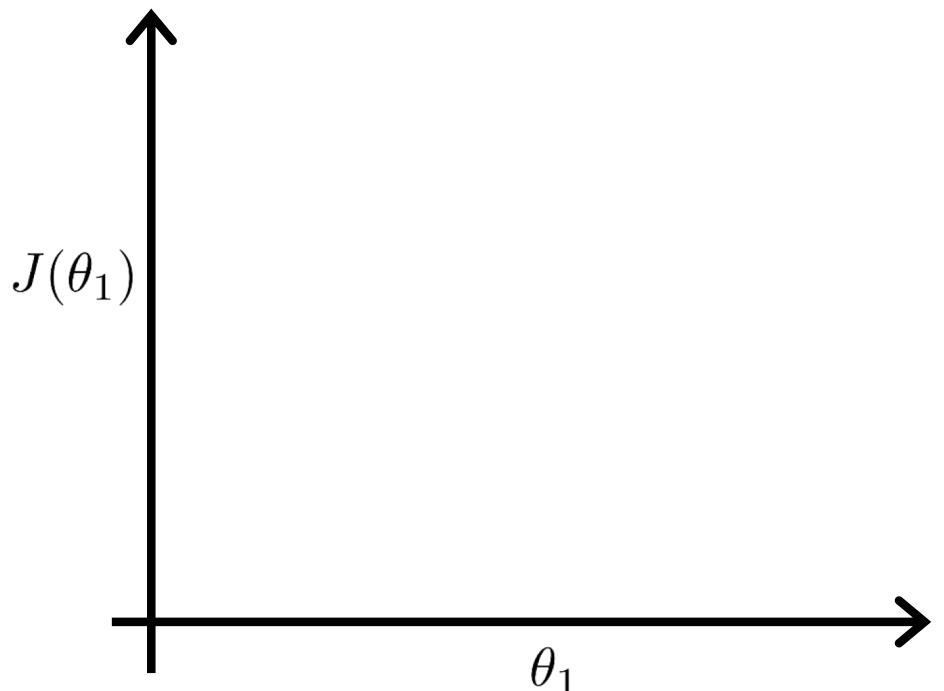


$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

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

As we approach a local minimum, gradient descent will automatically take smaller steps.



Gradient gets smaller as learning rate stays constant

Linear regression with one variable

Gradient descent algorithm

```
repeat until convergence {  
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$   
    (for  $j = 1$  and  $j = 0$ )  
}
```

Linear Regression Model

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

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

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

Finding steepest gradient of
Linear regression L2 cost function

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

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) =$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

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

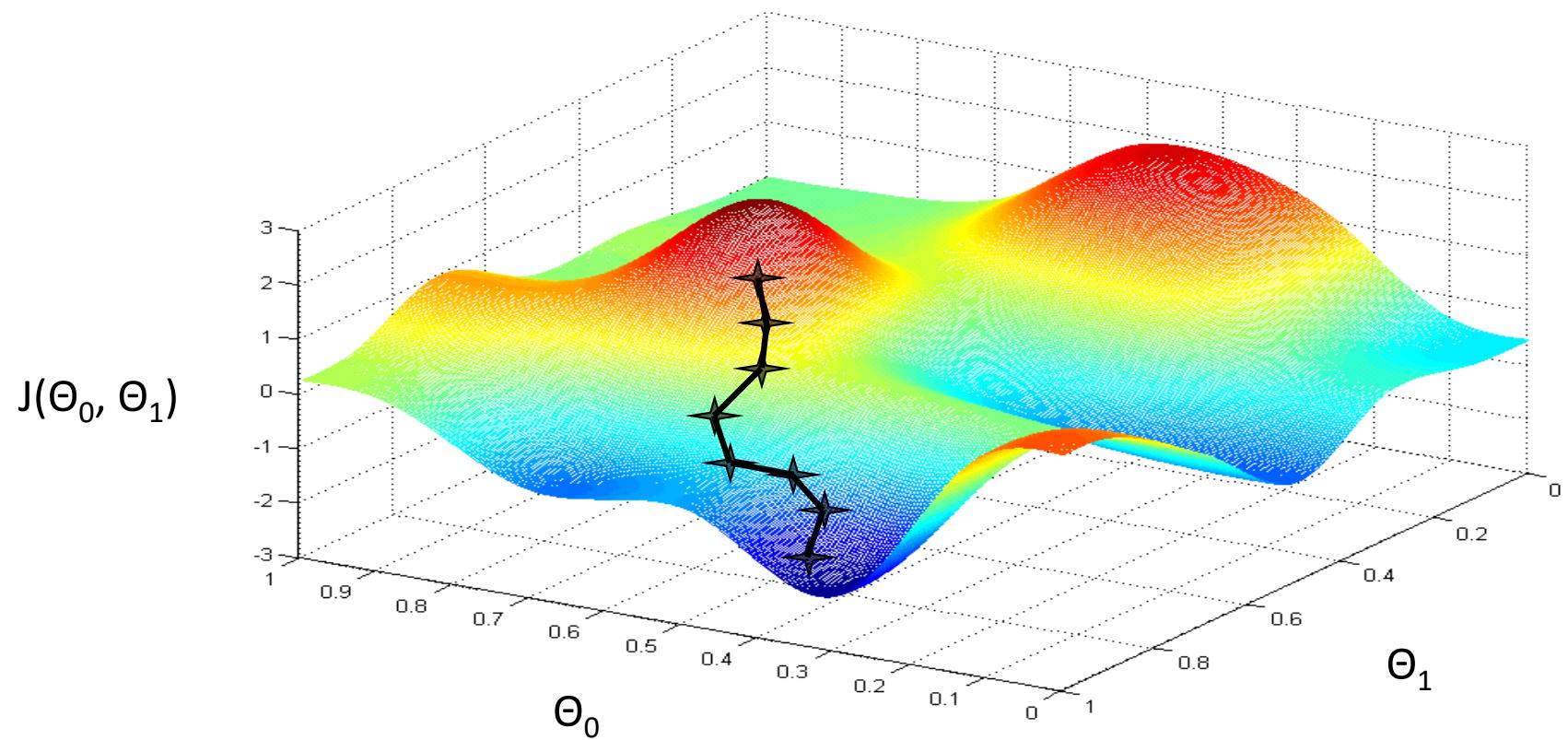
Gradient descent algorithm

repeat until convergence {

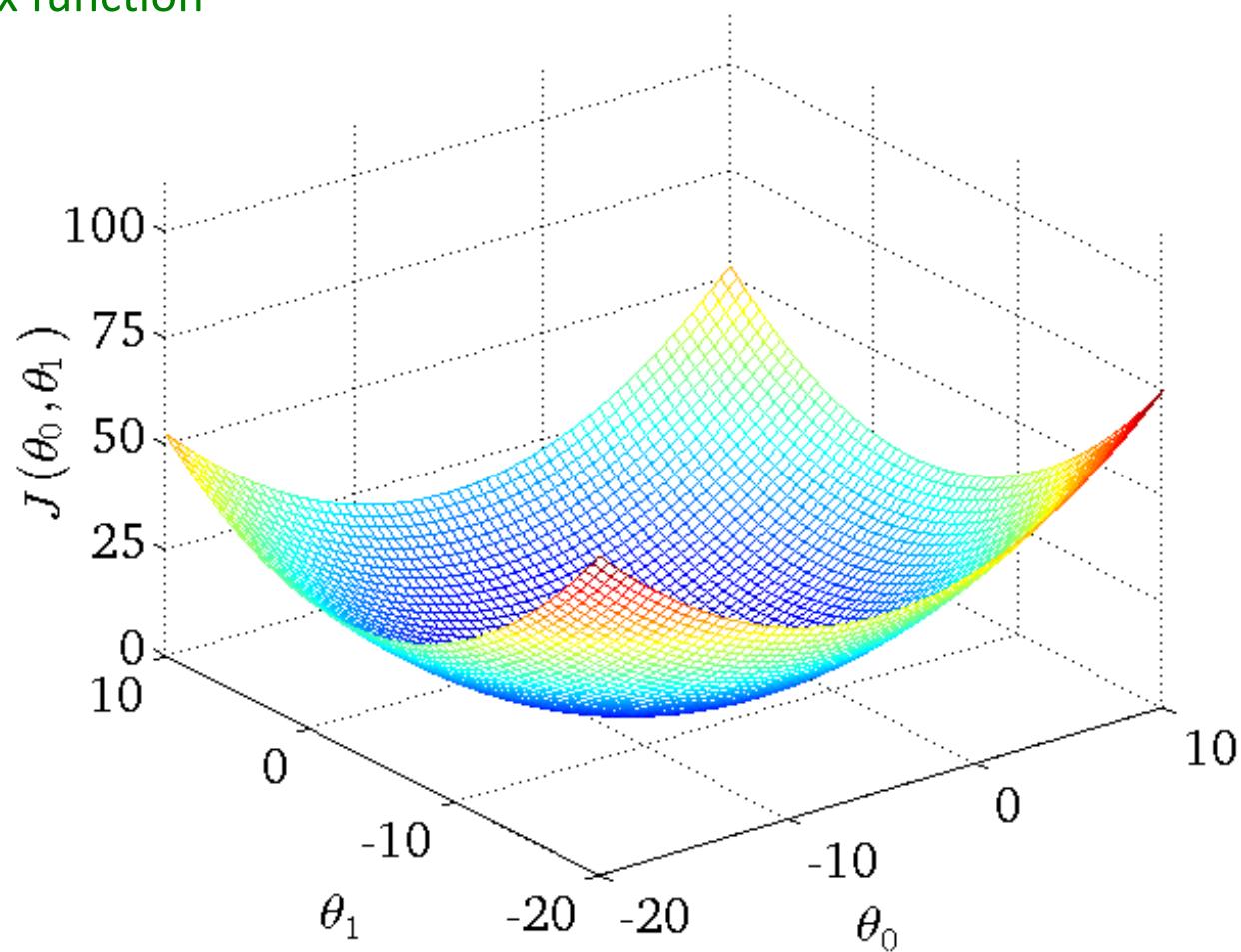
$$\left. \begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \end{aligned} \right\}$$

}

update
 θ_0 and θ_1
simultaneously



Convex function



“Batch” versus “Online” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

“Online”: Each step of gradient descent uses *a* training example at a time.

Multiple features (variables).

Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...

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

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

Cost function:

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

Gradient descent:

Repeat {

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

}

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

Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m \underbrace{(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

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

(simultaneously update θ_0, θ_1)

}

New algorithm ($n \geq 1$):

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
 $j = 0, \dots, n$)

}

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

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

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

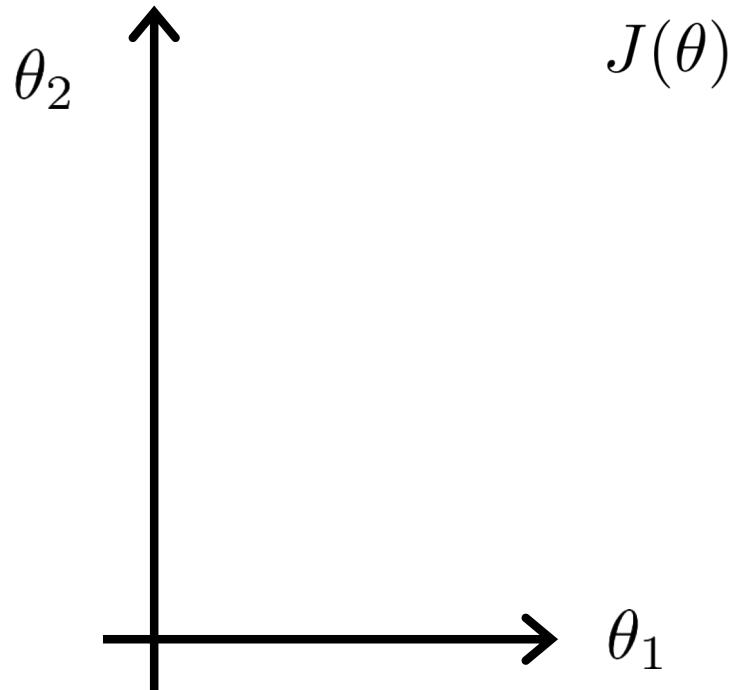
...

Feature Scaling

Idea: Make sure features are on a similar scale.

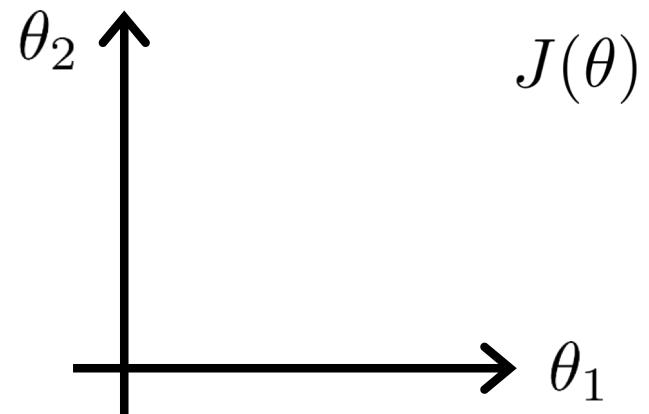
E.g. x_1 = size (0-2000 feet²)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean
(Do not apply to $x_0 = 1$).

E.g. $x_1 = \frac{\text{size} - 1000}{2000}$ $x_2 = \frac{\#\text{bedrooms} - 2}{5}$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

Z-score (standard score) normalization

$$z = \frac{x - \mu}{\sigma}$$
 Standard score is the signed number of standard deviations by which the value of an observation or data point is above the mean value.

Min-max (0-1) normalization

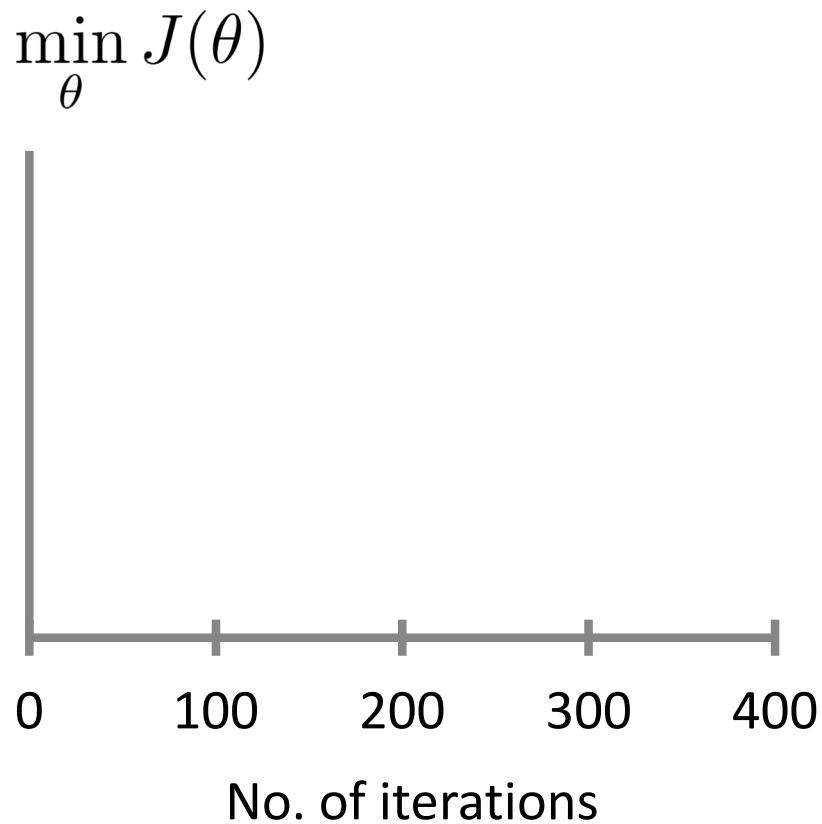
$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

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 .

Making sure gradient descent is working correctly.



Example automatic convergence test:

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

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

$\dots, 0.001, \quad , 0.01, \quad , 0.1, \quad , 1, \dots$

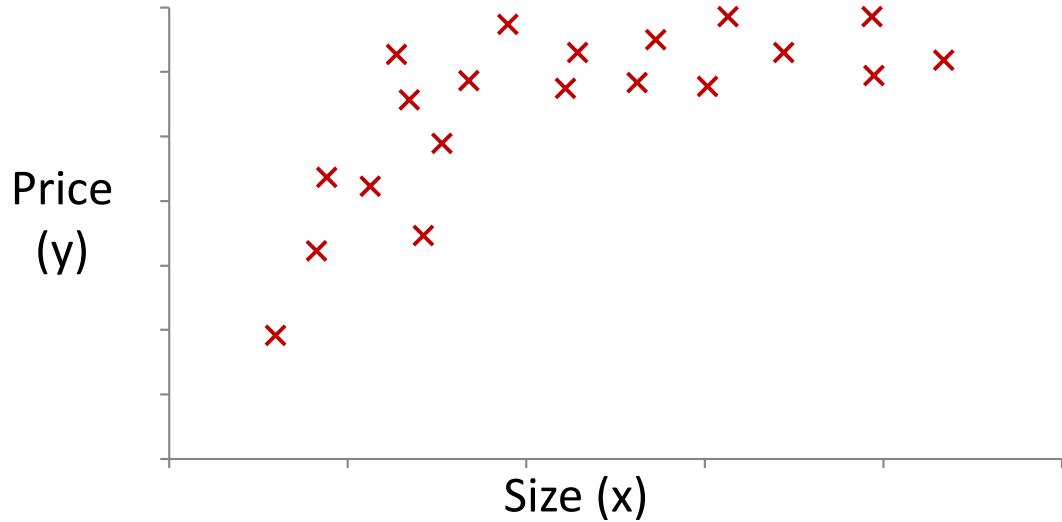
STOP FIRST LECTURE

Housing prices prediction

$$h_{\theta}(x) = \theta_0 + \theta_1 \times frontage + \theta_2 \times depth$$



Polynomial regression



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

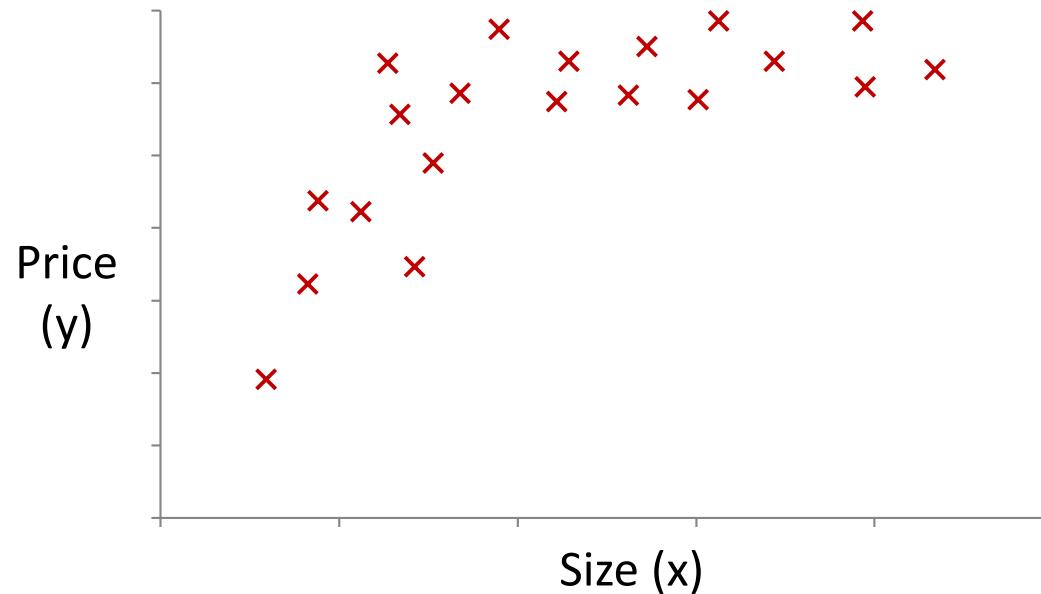
$$\begin{aligned}h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\&= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3\end{aligned}$$

$$x_1 = (\text{size})$$

$$x_2 = (\text{size})^2$$

$$x_3 = (\text{size})^3$$

Choice of features



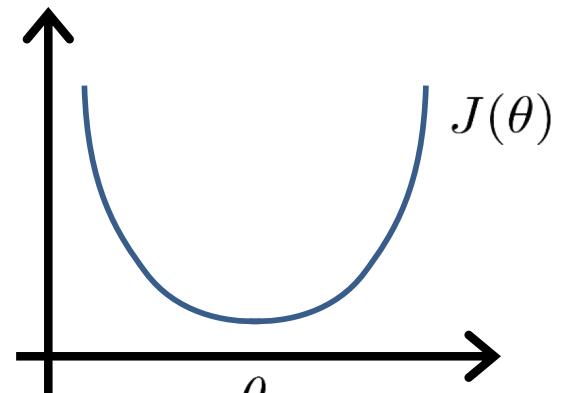
$$h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2$$

$$h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{(\text{size})}$$

Normal equation: Method to solve for analytically.

Intuition: If 1 parameter

$$J(\theta) = a\theta^2 + b\theta + c$$



$$(\theta \in \mathbb{R}) \qquad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \dots = 0 \quad (\text{for every } j)$$

Solve for $\theta_0, \theta_1, \dots, \theta_n$

Examples: $m = 5$.

	Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
x_0	x_1	x_2	x_3	x_4	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

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

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

Normal equation

[https://en.wikipedia.org/wiki/Linear_least_squares_\(mathematics\)](https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics))

Given the hypothesis function:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

We would like to minimize the least-squared error cost function:

Deriving the normal equation for linear regression

Jay Urbain

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

Representing the hypothesis function in matrix form, where θ_0 and x are vectors:

$$h_{\theta}(x) = \theta^T x$$

Updating the cost function to be in matrix form:

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

Dropping $\frac{1}{2m}$ since we're comparing a derivative of 0, and rearranging terms:

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

Note: Since $(X\theta)$ and y are vectors, we can multiply them in different orders provided the dimensions are correct.

$$J(\theta) = (X\theta)^T X\theta - 2(X\theta)^T y + y^T y$$

θ is what we are solving for, to find a minimum for our cost function $J(\theta)$, we need to take the derivatives of J with respect to θ .

$$\frac{\partial J}{\partial \theta} = 2X^T X\theta - 2X^T y = 0$$

or

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

Multiply both sides by $(X^T X)^{-1}$

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

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

Show derivation notebook

$(X^T X)^{-1}$ is inverse of matrix $X^T X$.

`numpy.linalg.solve(a, b)`

<http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.linalg.solve.html>

theta = np.linalg.solve(X, y)

Assumes X is not singular, and is square

Matlab/Octave: **pinv(X' *X) *X' *y**

m training examples, n features.

Gradient Descent

- Need to choose α .
- Needs many iterations.
- Works well even when n is large.

Normal Equation

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

What if $X^T X$ is non-invertible?

- Redundant features (linearly dependent).
E.g. $x_1 = \text{size in feet}^2$
 $x_2 = \text{size in m}^2$
- Too many features (e.g. $m \leq n$).
 - Delete some features, or use regularization.

Problem: considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities.

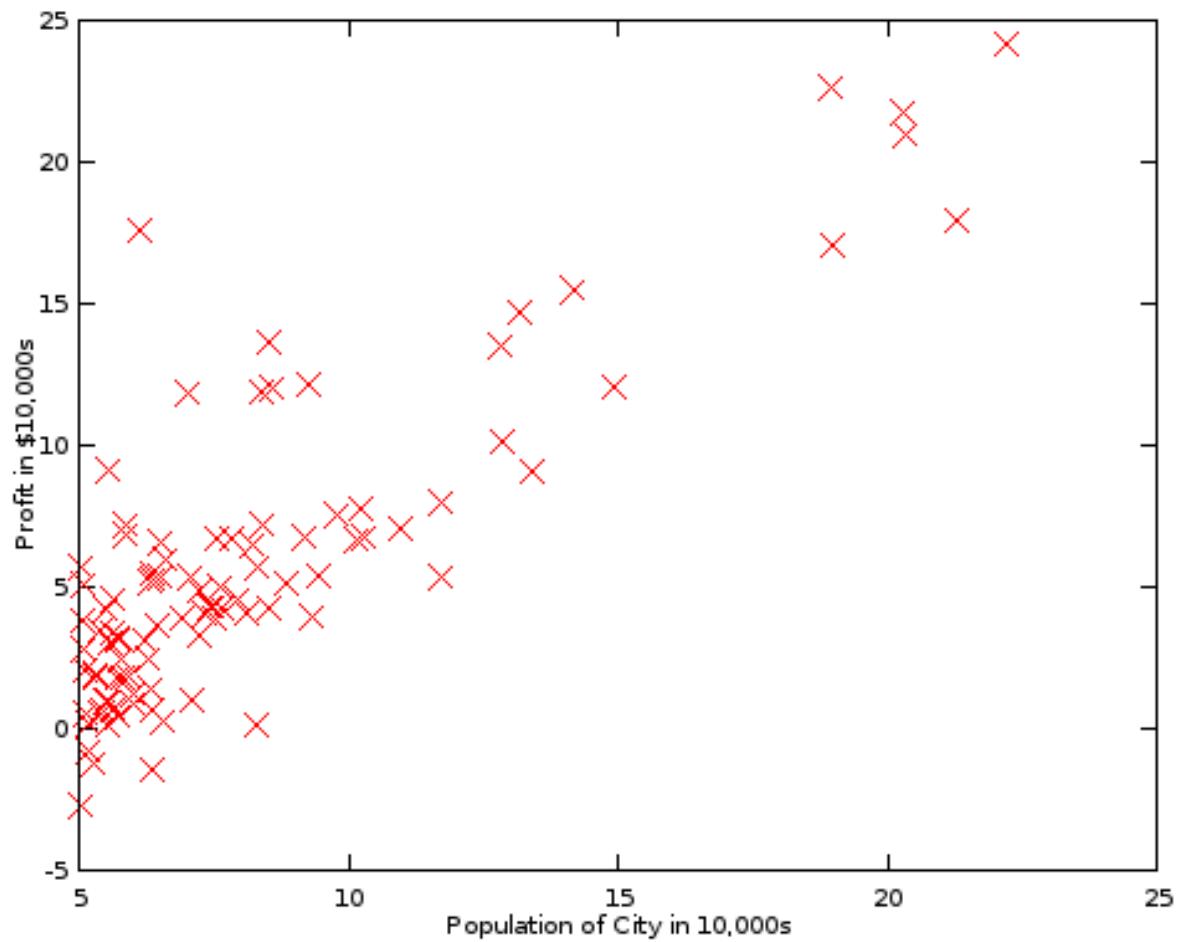
You would like to use this data to help you select which city to expand to next.

X =

y =

1.0000	6.1101	17.59200
1.0000	5.5277	9.13020
1.0000	8.5186	13.66200
1.0000	7.0032	11.85400
1.0000	5.8598	6.82330
1.0000	8.3829	11.88600
1.0000	7.4764	4.34830
1.0000	8.5781	12.00000
1.0000	6.4862	6.59870
1.0000	5.0546	3.81660
1.0000	5.7107	3.25220
1.0000	14.1640	15.50500
1.0000	5.7340	3.15510

Visualize data

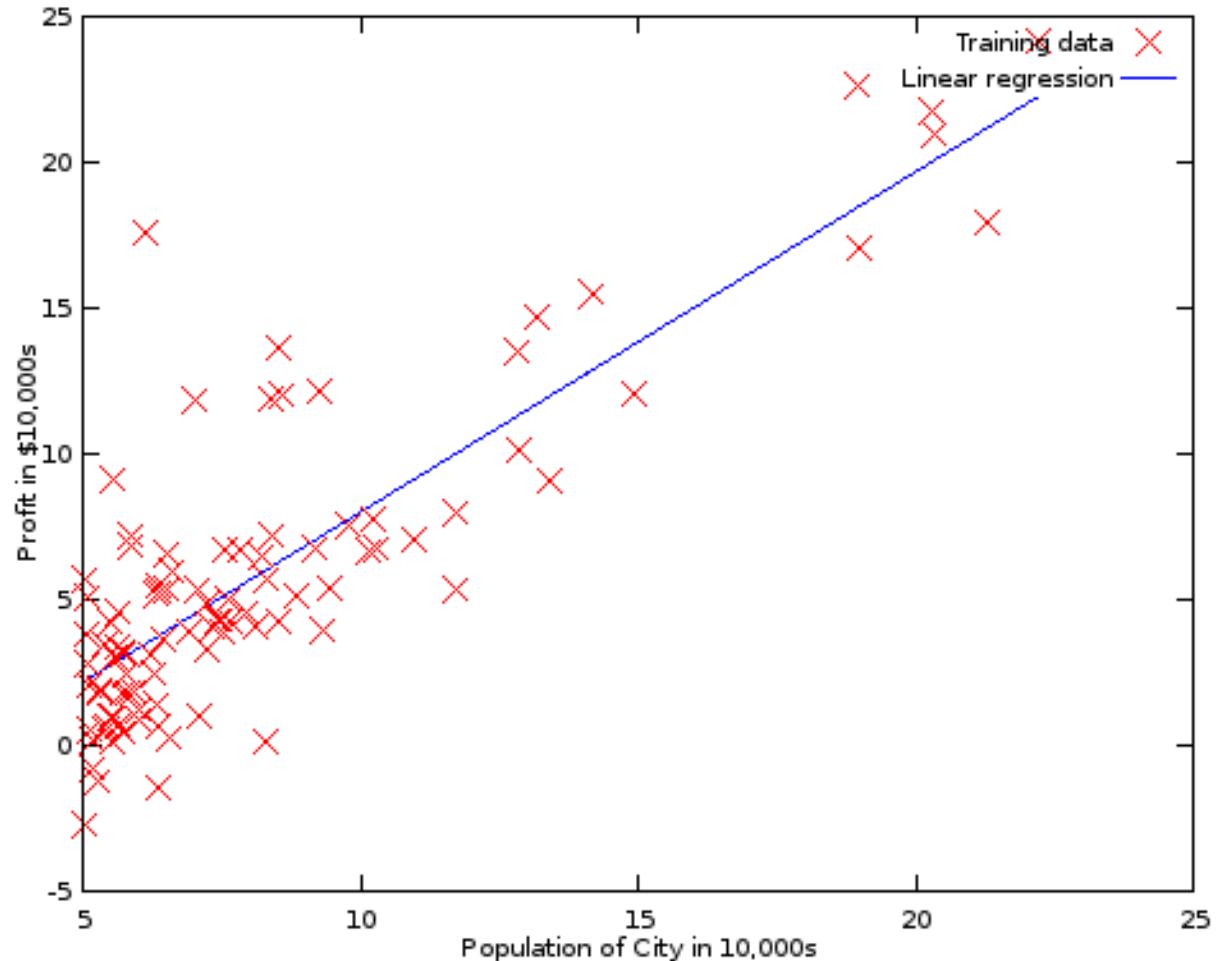


Run linear regression

Theta found by gradient
descent: -3.630291 1.166362

For population = 35,000, we
predict a profit of
4519.767868

For population = 70,000, we
predict a profit of
45342.450129



Assessing the accuracy of model coefficients

Linear regression with residual term. Represents what we can't explain with our model.

RSS measures the amount of variability that is left unexplained after performing the regression

TSS (Total sum of squares) measures the total variance when measuring the response y .

R^2 amount of variance explained by our model

The RSE is an estimate of the standard deviation of ϵ . It is basically the average amount that the response will deviate from the true regression line.

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

$$\text{TSS} = \sum (y_i - \bar{y})^2$$

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

$$\text{RSE} = \sqrt{\frac{1}{n-2} \text{RSS}} = \sqrt{\frac{1}{n-2} \sum_{i=1}^n (y_i - \hat{y}_i)^2}.$$

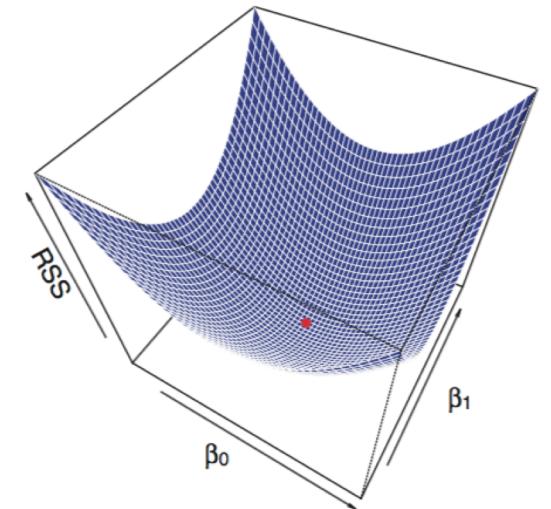
Least square approach

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

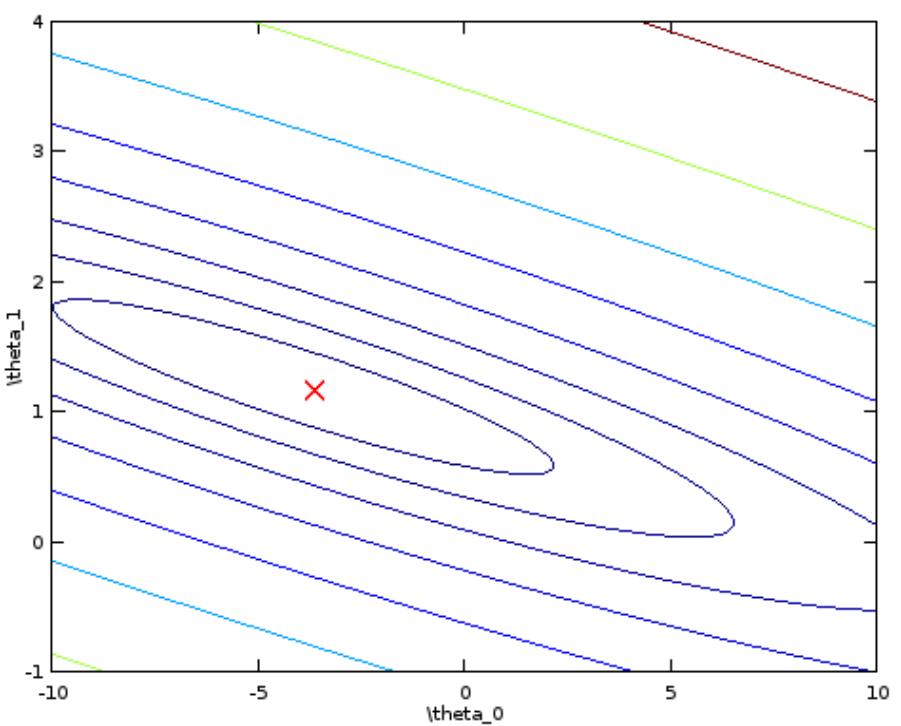
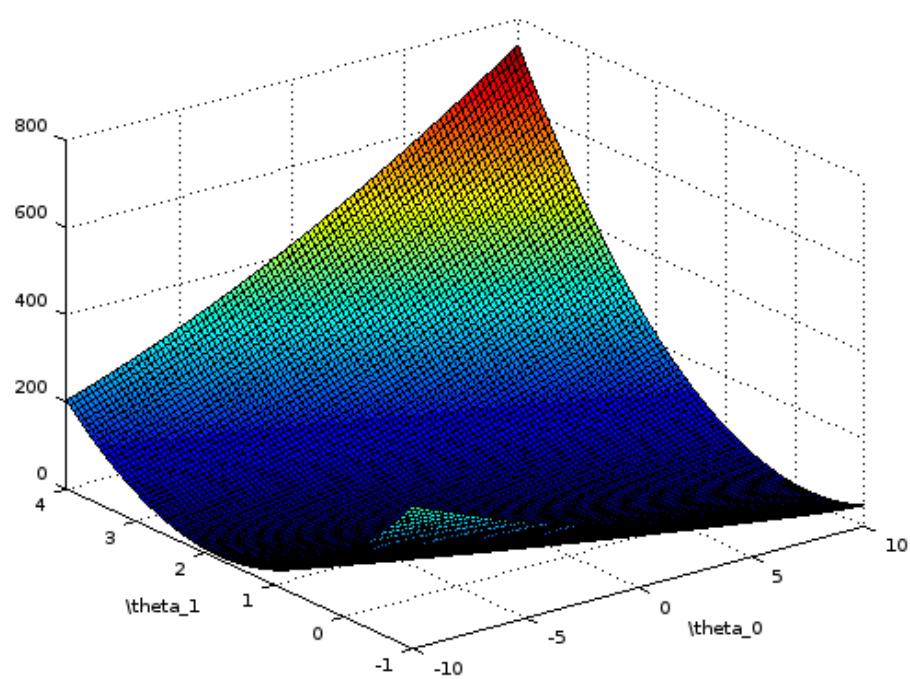
$$\text{RSS} = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \dots + (y_n - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2.$$

The least squares approach chooses β_0 and β_1 to minimize the RSS. Using some calculus, one can show that the minimizers are:



$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2},$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

Saving and plotting theta



Notes on:

- Feature Scaling
- Nonlinear Regression
- Optimizing Cost using derivatives

Demo gradient descent algorithm
-> 08_linear_regression.ipynb

STOP LECTURE 2