DS 4400

Machine Learning and Data Mining I

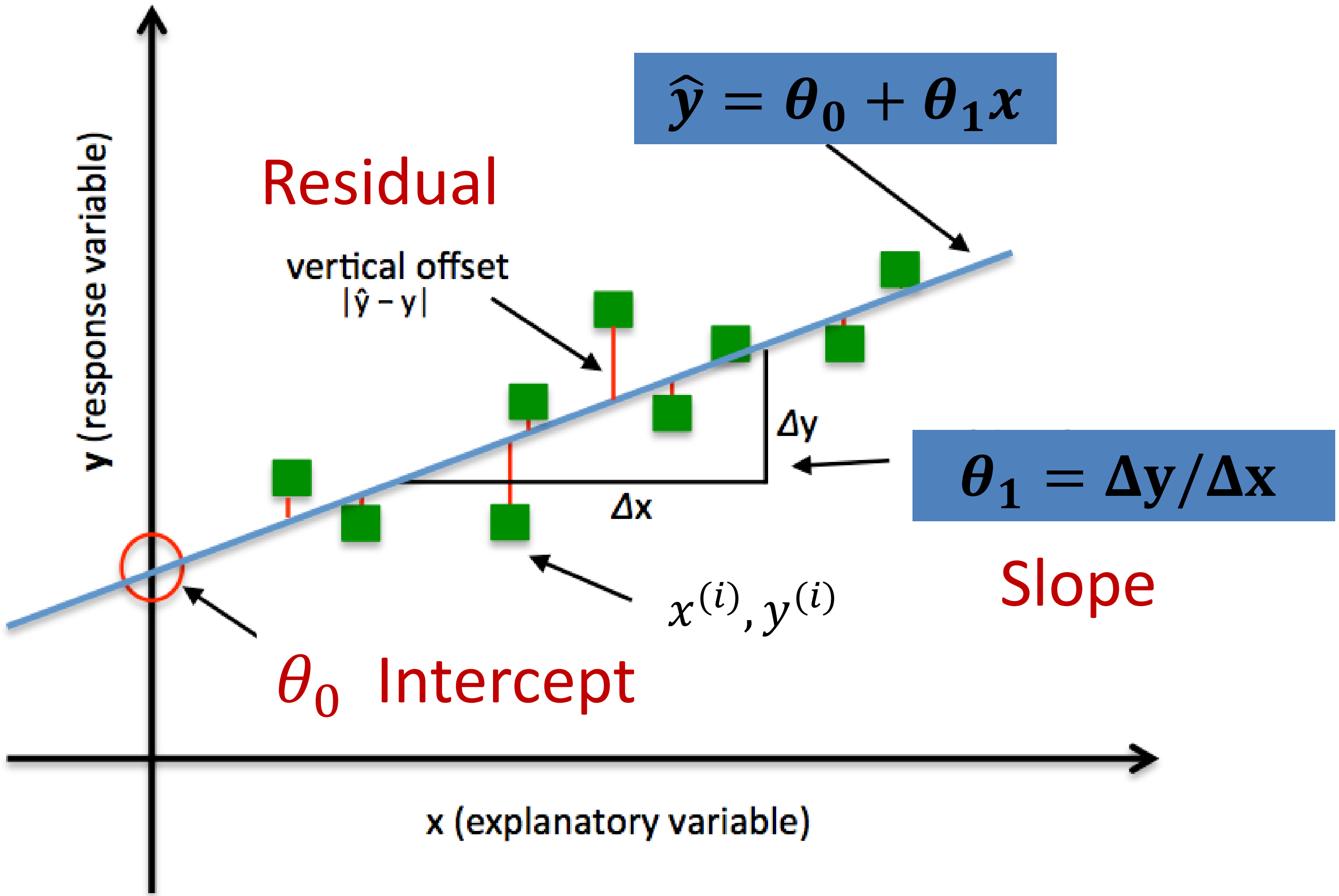
Alina Oprea
Associate Professor, CCIS
Northeastern University

January 22 2019

Outline

- Practical issues in Linear Regression
  - Outliers
  - Categorical variables
- Lab Linear Regression
- Gradient descent
  - Efficient algorithm for optimizing loss function
  - Training Linear Regression with Gradient Descent
  - Comparison with closed-form solution

Simple Linear Regression



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

Loss: MSE$= \frac{1}{n} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Simple Linear Regression

- Dataset $x^{(i)} \in R, y^{(i)} \in R, h_\theta(x) = \theta_0 + \theta_1 x$
- $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$

  MSE / Loss

- Solution of min loss

$$- \theta_0 = \bar{y} - \theta_1 \bar{x}$$

$$- \theta_1 = \frac{\sum (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sum \left(x^{(i)} - \bar{x}\right)^2}$$

$$\bar{x} = \frac{\sum_{i=1}^{n} x^{(i)}}{n}$$

$$\bar{y} = \frac{\sum_{i=1}^{n} y^{(i)}}{n}$$

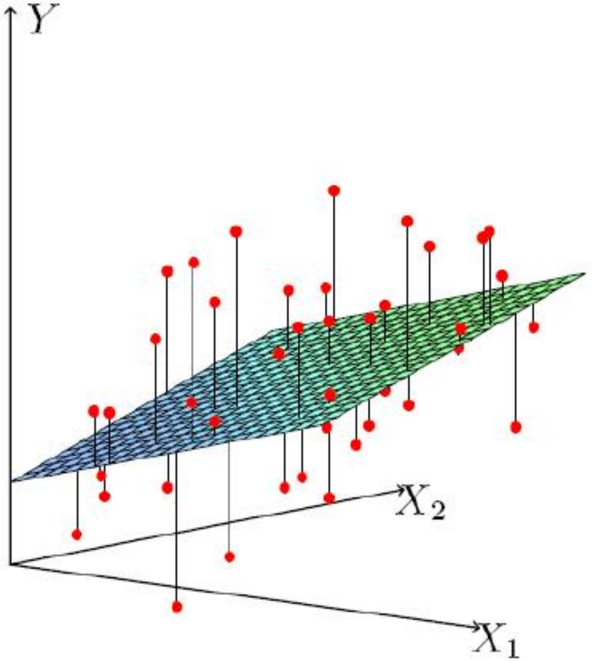
Variance of x (denominator of θ_1)

Co-variance of x and y (numerator of θ_1)

Multiple Linear Regression

- Dataset: $x^{(i)} \in R^d, y^{(i)} \in R$
- Hypothesis $h_\theta(x) = \theta^T x$
- MSE $= \frac{1}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} - y^{(i)}\right)^2$ Loss / cost

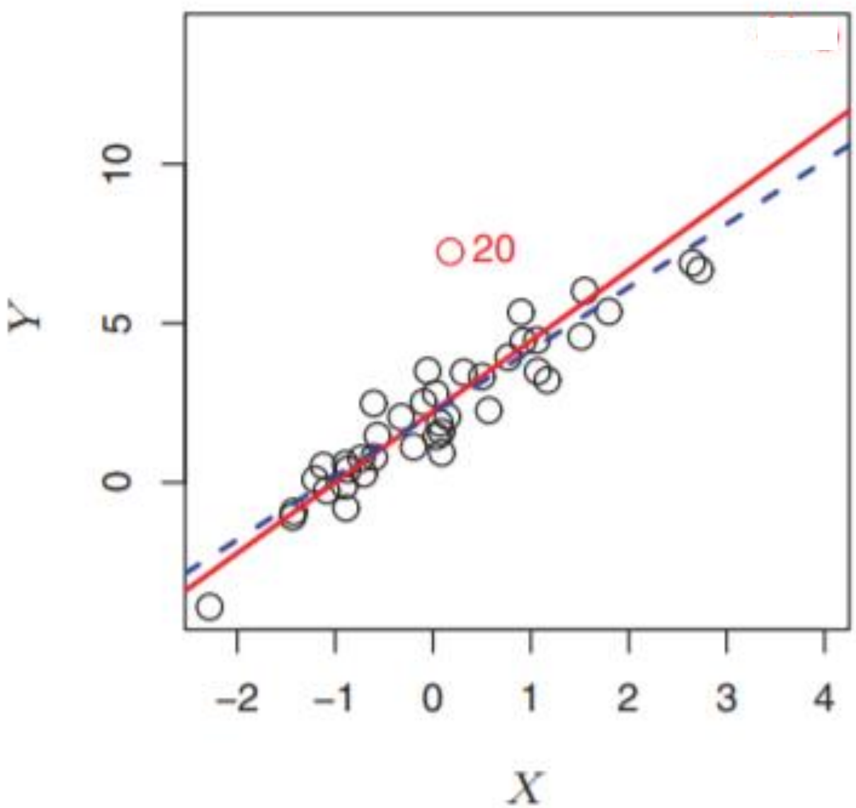
$$\theta = (X^\top X)^{-1} X^\top y$$

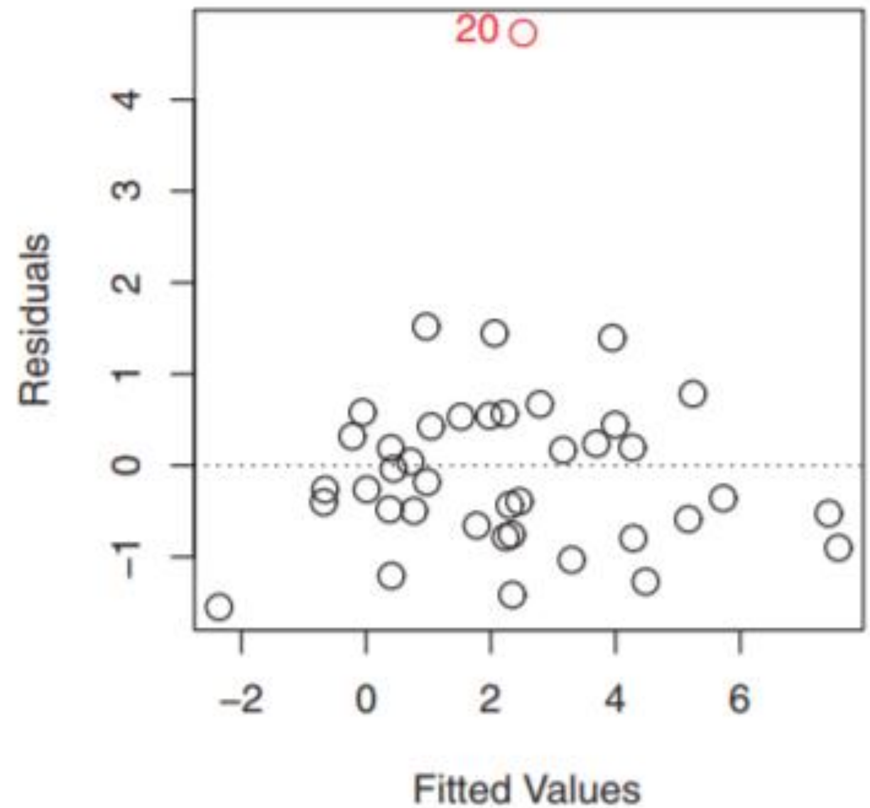


Feature standardization/normalization

- Goal is to have individual features on the same scale
- Is a pre-processing step in most learning algorithms
- Necessary for linear models and Gradient Descent
- Different options:
  - Feature standardization
  - Feature min-max rescaling
  - Mean normalization

Outliers





- Dashed model is without outlier point
- Linear regression is not resilient to outliers!
- Outliers can be eliminated based on residual value
  - Other techniques for outlier detection

Categorical variables

- Predict credit card balance
  - Age
  - Income
  - Number of cards
  - Credit limit
  - Credit rating
- Categorical variables
  - Student (Yes/No)
  - State (50 different levels)

Indicator Variables

- Binary (two-level) variable
  - Add new feature $x_j = 1 \text{ if student and } 0 \text{ otherwise}$
- Multi-level variable
  - State: 50 values
  - $x_{MA} = 1 \text{ if State} = \text{MA and } 0, \text{otherwise}$
  - $x_{NY} = 1 \text{ if State} = \text{NY and } 0, \text{otherwise}$
  - …
  - How many indicator variables are needed?
- Disadvantages: data becomes too sparse for large number of levels
  - Will discuss feature selection later in class

Lab example

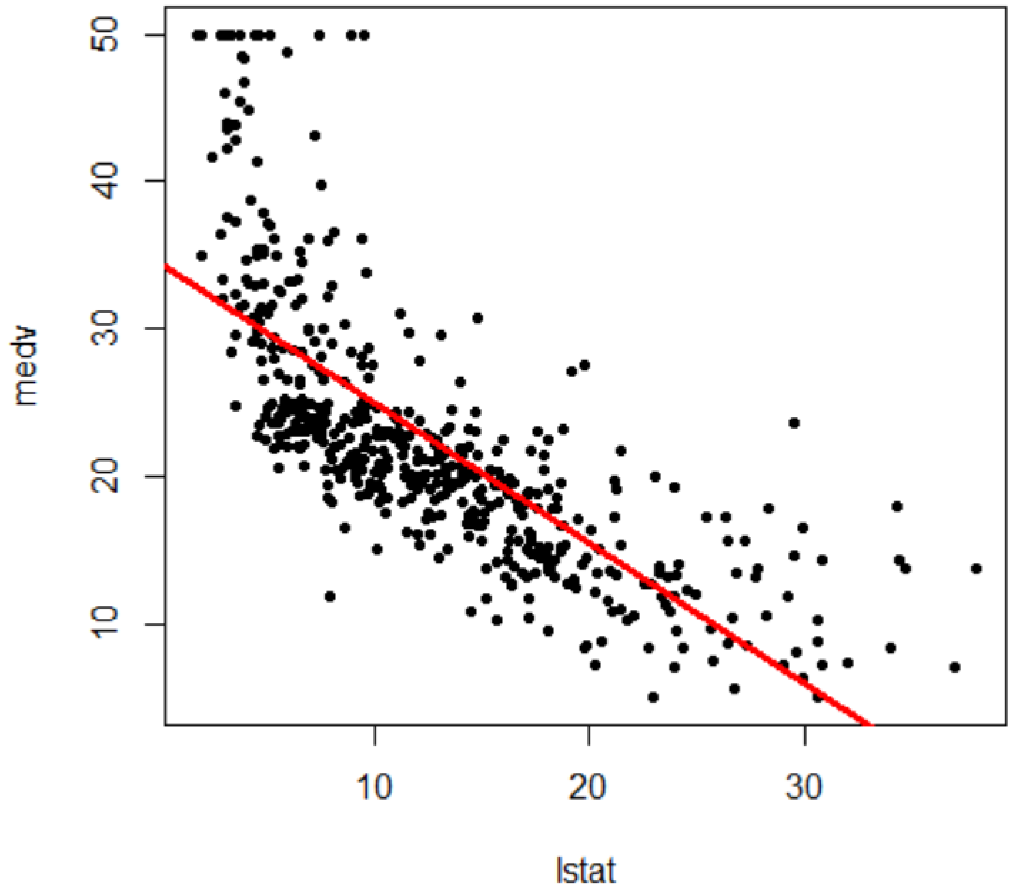
```
>
> library(MASS)
> fix(Boston)
> |
```

Data Editor

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
1	0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
2	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
3	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
4	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
5	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
6	0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
7	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9
8	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1
9	0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5
10	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9
11	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15
12	0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9
13	0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7
14	0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4
15	0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2
16	0.62739	0	8.14	0	0.538	5.834	56.5	4.4986	4	307	21	395.62	8.47	19.9

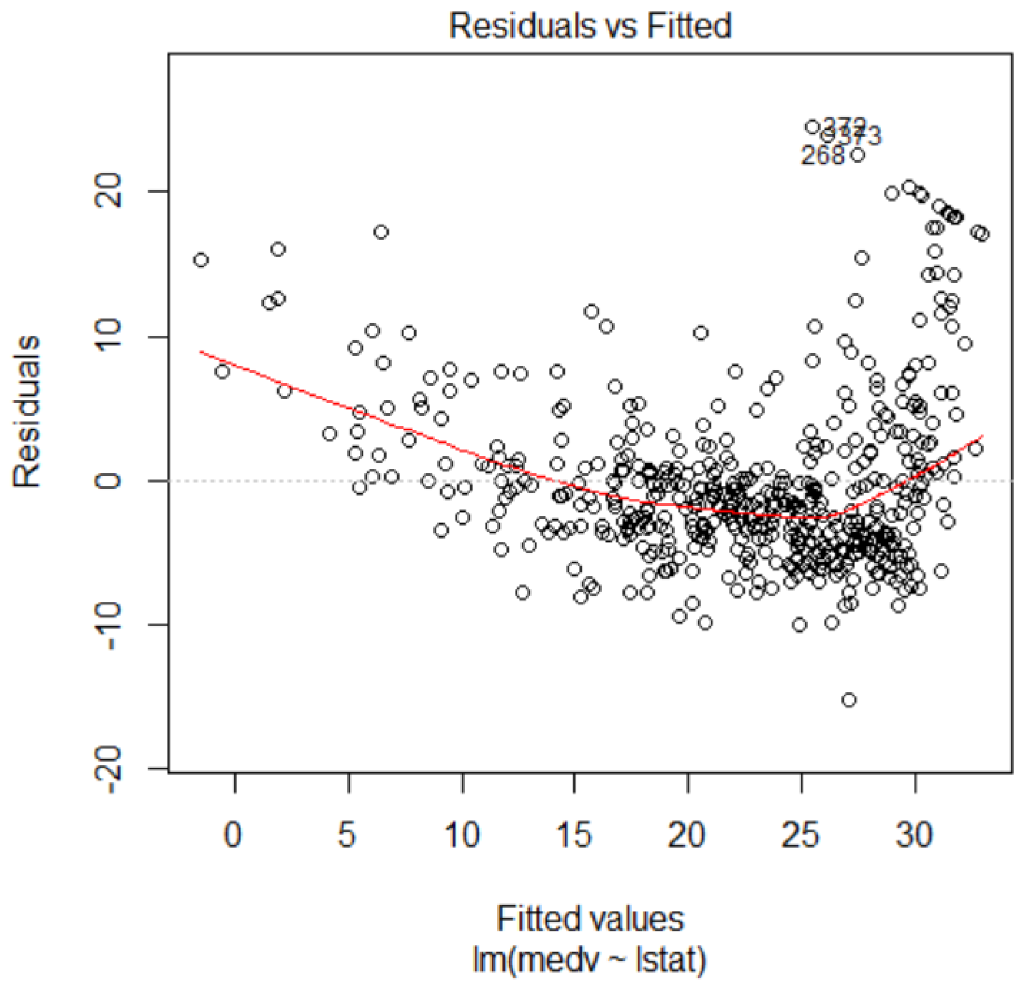
Simple LR

```
>
> lm.fit=lm(medv~lstat,data=Boston)
> plot(lstat,medv,pch=20)
> abline(lm.fit,lwd=3,col="red")
>
```



Residual plot

```
>
> plot(predict(lm.fit), residuals(lm.fit))
>
> plot(lm.fit, which=1)
>
```



Estimated responses

Simple LR

```
>
> lm.fit=lm(medv~lstat,data=Boston)
> summary(lm.fit)

Call:
lm(formula = medv ~ lstat, data = Boston)

Residuals:
    Min      1Q  Median      3Q     Max
-15.168  -3.990  -1.318   2.034  24.500

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 34.55384    0.56263   61.41   <2e-16 ***
lstat       -0.95005    0.03873  -24.53   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.216 on 504 degrees of freedom
Multiple R-squared:  0.5441,    Adjusted R-squared:  0.5432
F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

Coef not zero!

$$RSE = \sqrt{MSE}$$

R^2 measures linear relationship between X and Y (equal to correlation coef for simple LR)

Multiple LR

```
> lm.fit=lm(medv~nox+rm+lstat+ptratio+rad+dis,data=Boston)
> summary(lm.fit)

Call:
lm(formula = medv ~ nox + rm + lstat + ptratio + rad + dis, d$

Residuals:
     Min       1Q   Median       3Q      Max 
-12.8663  -3.1525  -0.5509   1.9870  27.1748 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  40.61722    5.07480   8.004 8.53e-15 ***
nox         -20.16431    3.57710  -5.637 2.90e-08 ***
rm            4.04507    0.41938   9.645  < 2e-16 ***
lstat        -0.59197    0.04846 -12.217  < 2e-16 ***
ptratio      -1.12748    0.12634  -8.924  < 2e-16 ***
rad           0.05399    0.03682   1.466    0.143    
dis          -1.19580    0.16840  -7.101 4.29e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.988 on 499 degrees of freedom
Multiple R-squared:  0.7093,    Adjusted R-squared:  0.7058 
F-statistic:   203 on 6 and 499 DF,  p-value: < 2.2e-16
```

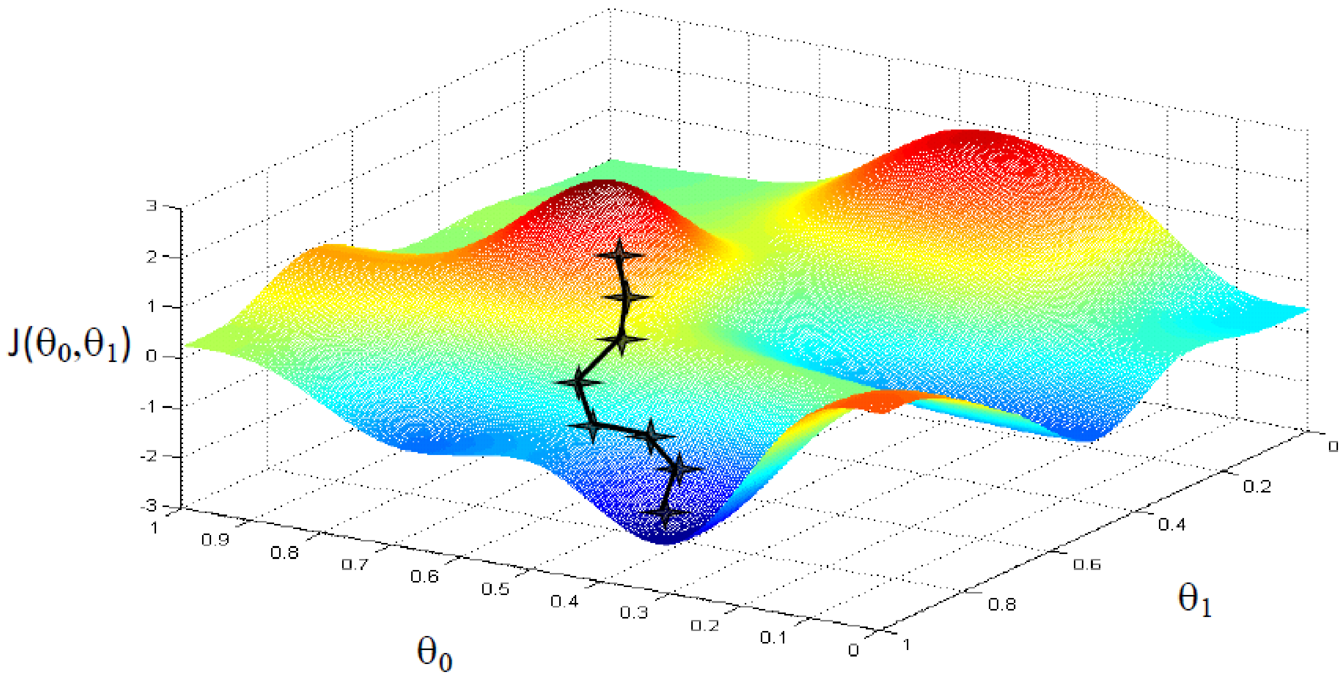
What Strategy to Use?

Follow the Slope

Follow the direction of steepest descent!

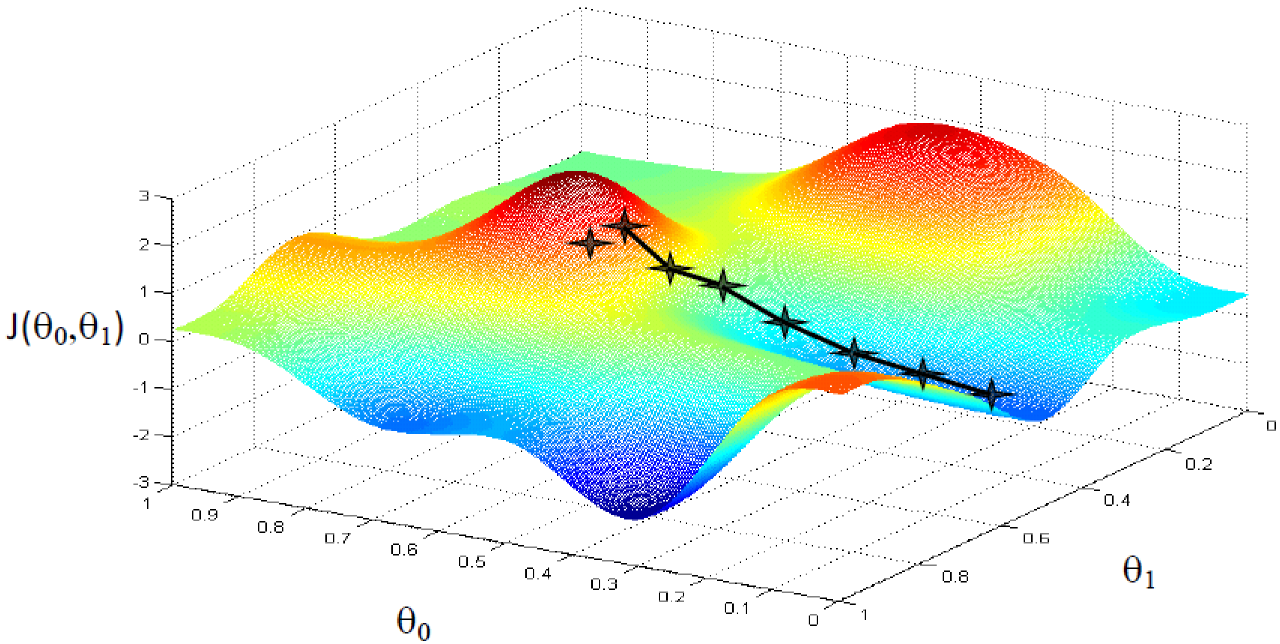
How to optimize $J(\theta)$?

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for θ to reduce $J(\theta)$



How to optimize $J(\theta)$?

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for $\boldsymbol{\theta}$ to reduce $J(\boldsymbol{\theta})$



Different starting point

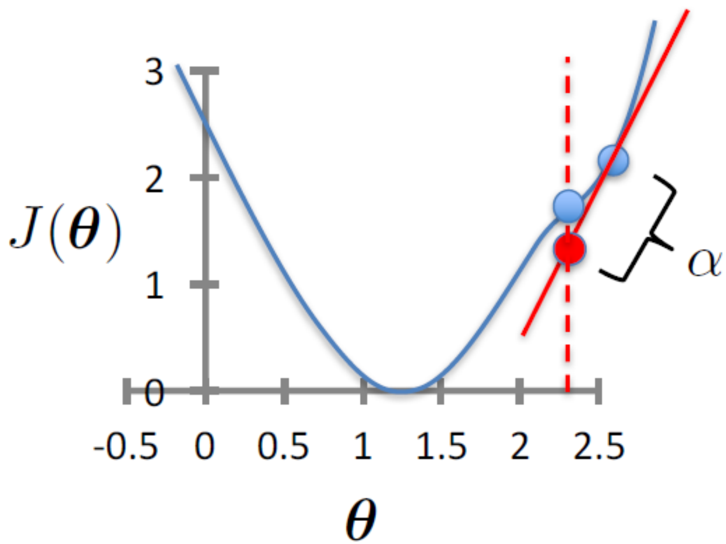
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

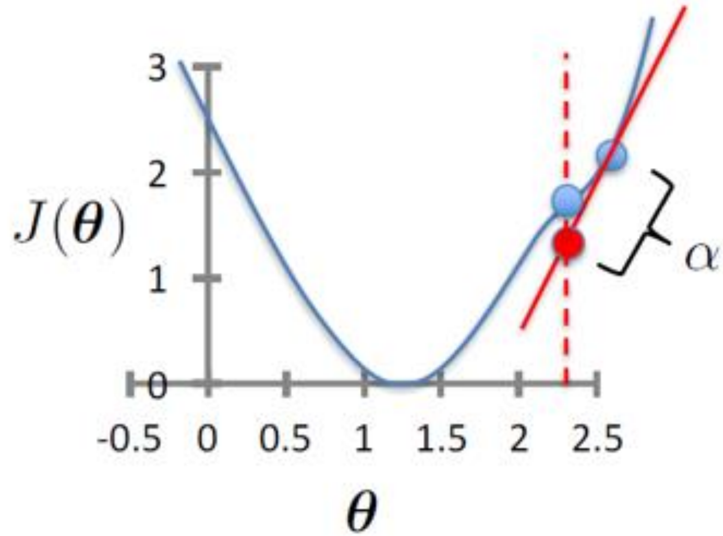
simultaneous update
for j = 0 ... d

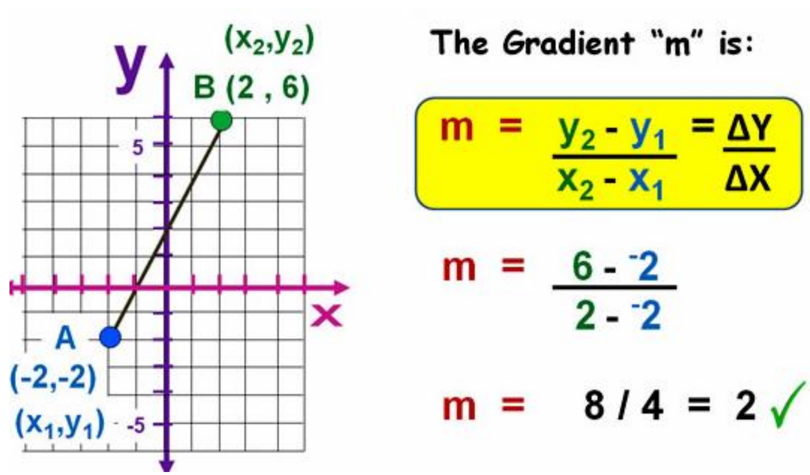
learning rate (small)
e.g., α = 0.05



- Gradient = slope of line tangent to curve
- Function decreases faster in negative direction of gradient
- Larger learning rate => larger step

Gradient Descent





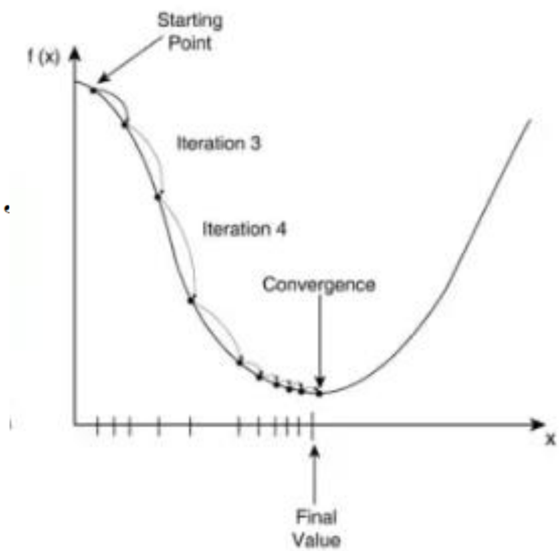
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



- As you approach the minimum, the slope gets smaller, and GD will take smaller steps
- It converges to local minimum (which is global minimum for convex functions)!

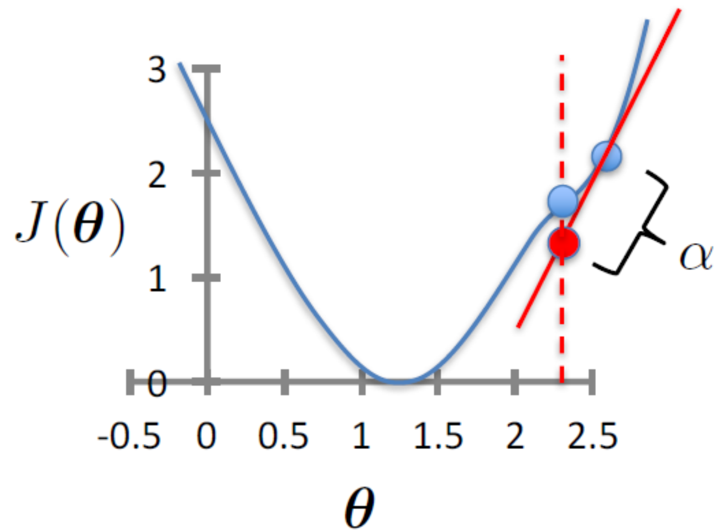
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

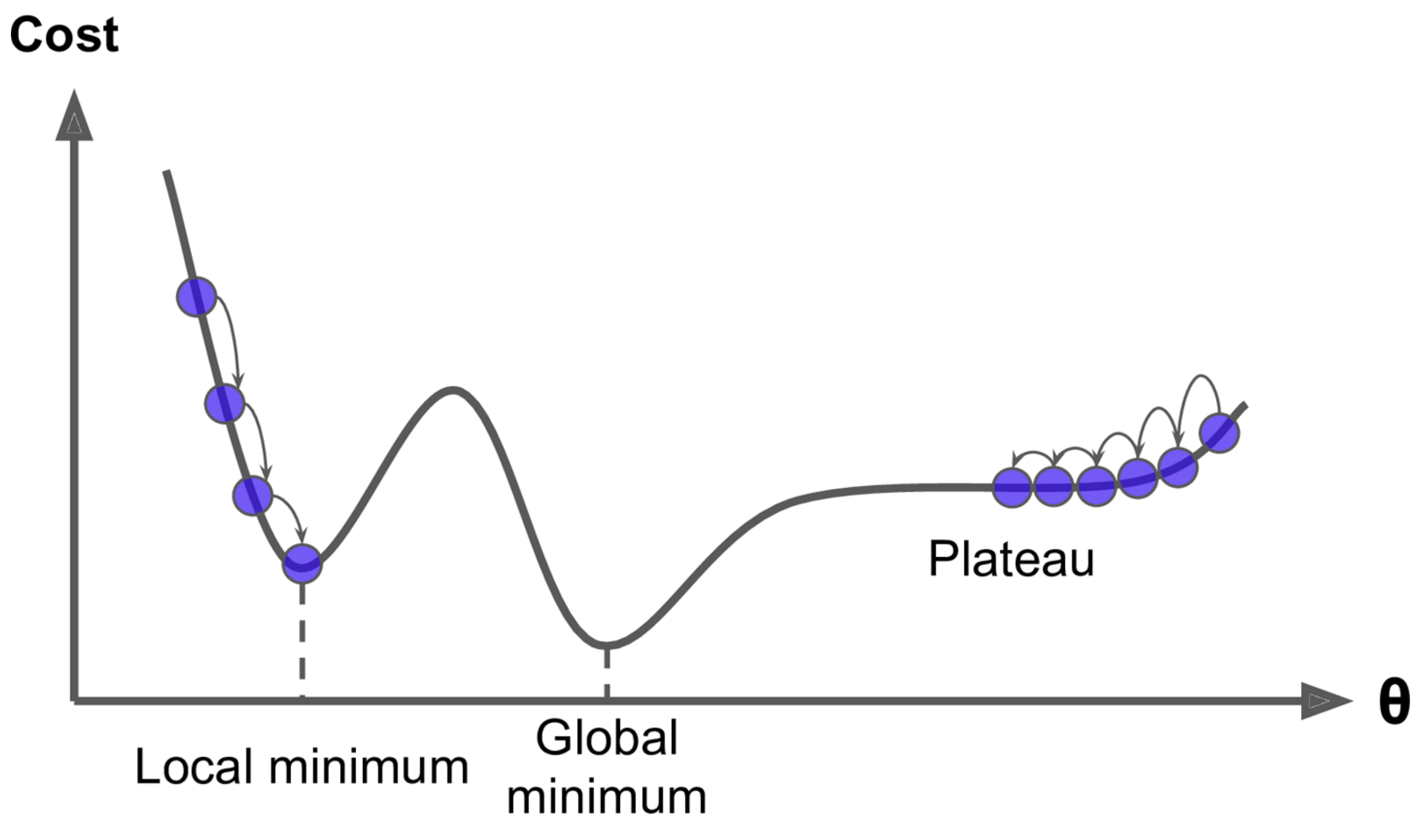
simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



- What happens when θ reaches a local minimum?
- The slope is 0, and gradient descent converges!

GD Converges to Local Minimum



Solution: start from multiple random locations

GD for Simple Linear Regression

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

- $J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$
- $\frac{\partial J(\theta)}{\partial \theta_0} = \frac{2}{n} \sum_{i=1}^{n} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$
- $\frac{\partial J(\theta)}{\partial \theta_1} = \frac{2}{n} \sum_{i=1}^{n} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})\, x^{(i)}$

Update of each parameter component depends on all training data

GD for Multiple Linear Regression

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update for j = 0 ... d

For Linear Regression:
$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_j} \frac{1}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)^2$$

$$= \frac{\partial}{\partial \theta_j} \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)^2$$

$$= \frac{2}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)$$

$$= \frac{2}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}$$

GD for Linear Regression

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$||\theta_{new} - \theta_{old}|| < \epsilon$ or
iterations == MAX_ITER

$$\theta_j \leftarrow \theta_j - \alpha \frac{2}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

simultaneous update for j = 0 ... d

- To achieve simultaneous update
  - At the start of each GD iteration, compute $h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)$
  - Use this stored value in the update step loop
- Assume convergence when $\|\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old}\|_2 < \epsilon$

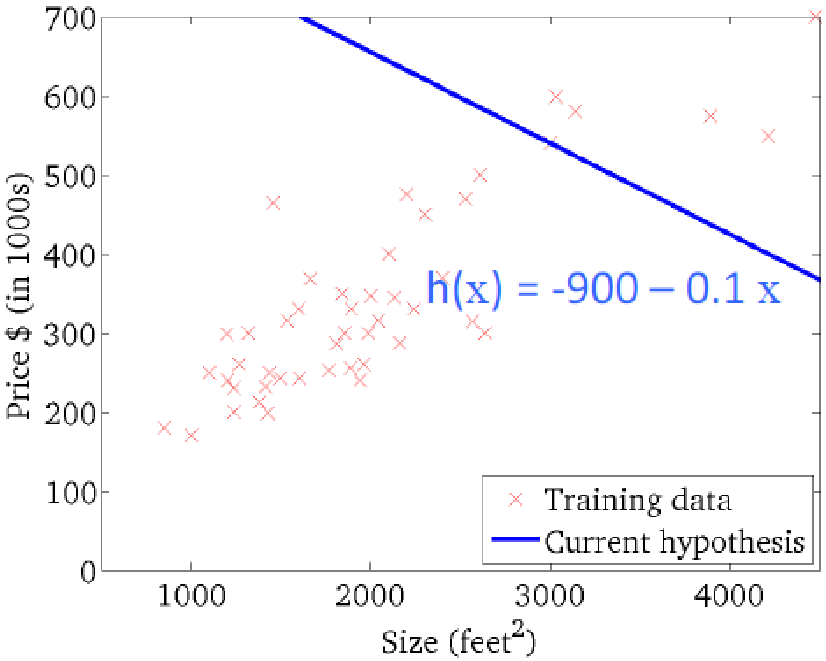
L_2 norm: $$\|\boldsymbol{v}\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \ldots + v_{|v|}^2}$$

Can also bound number of iterations

GD Example

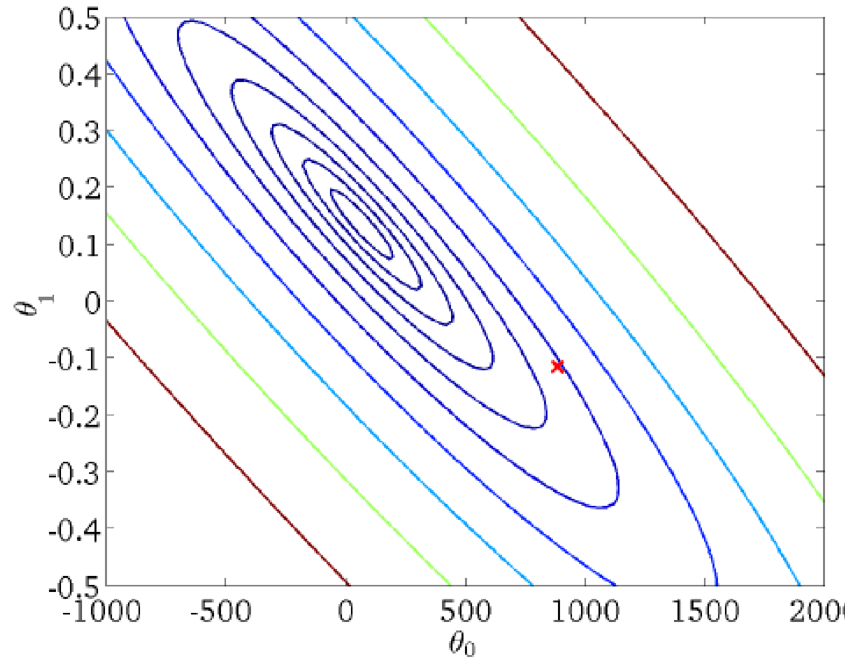
$$h_\theta(x)$$

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



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

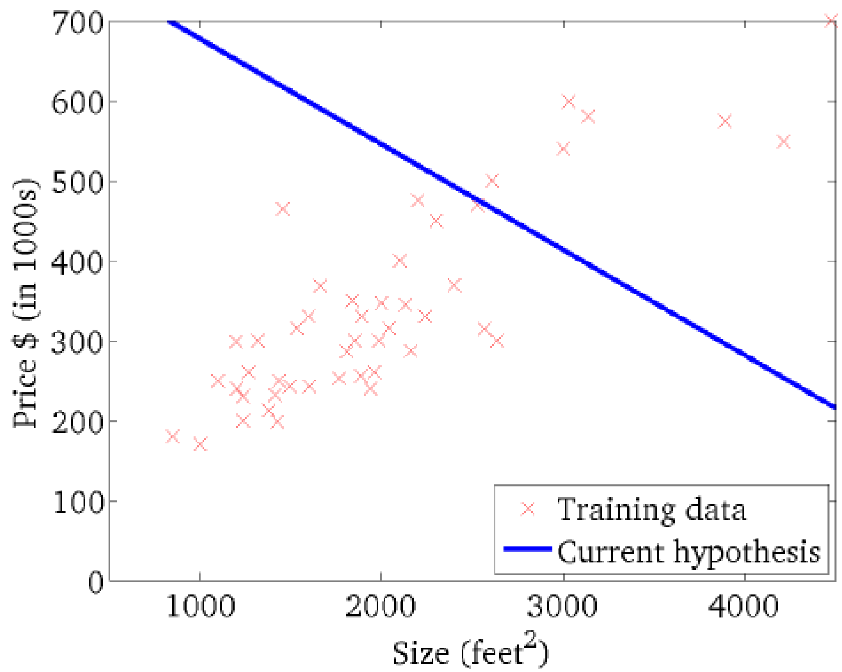
(function of the parameters θ_0, θ_1)



GD Example

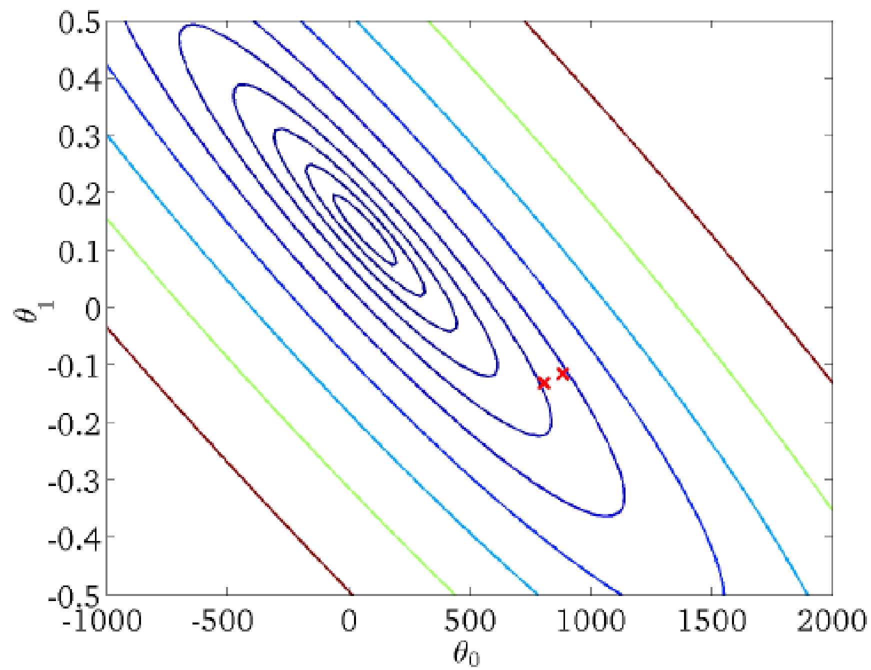
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

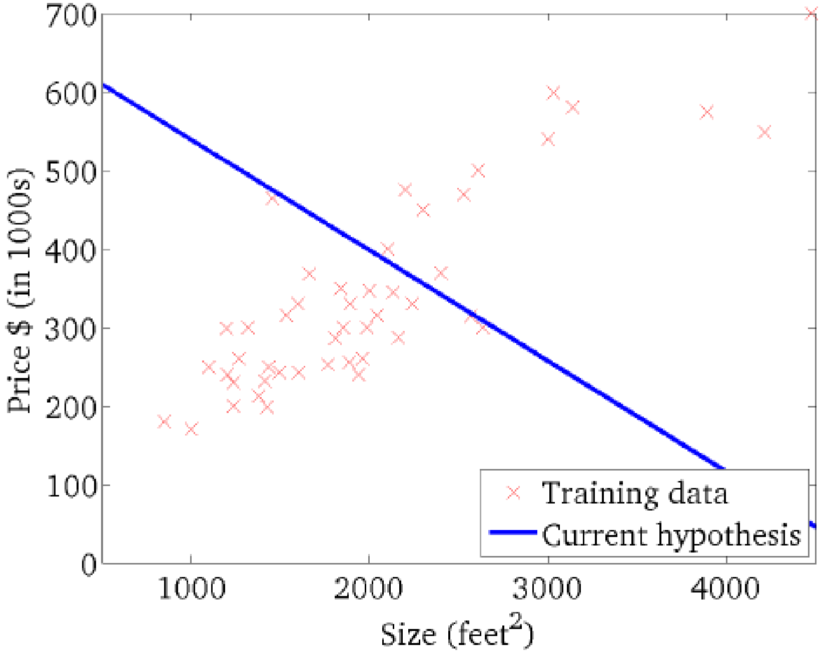
(function of the parameters θ_0, θ_1)



GD Example

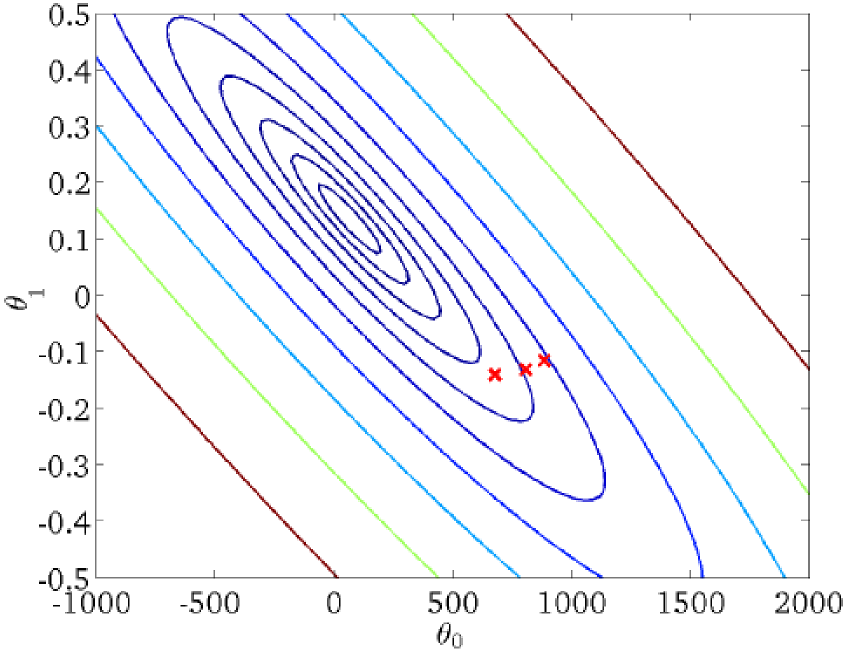
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

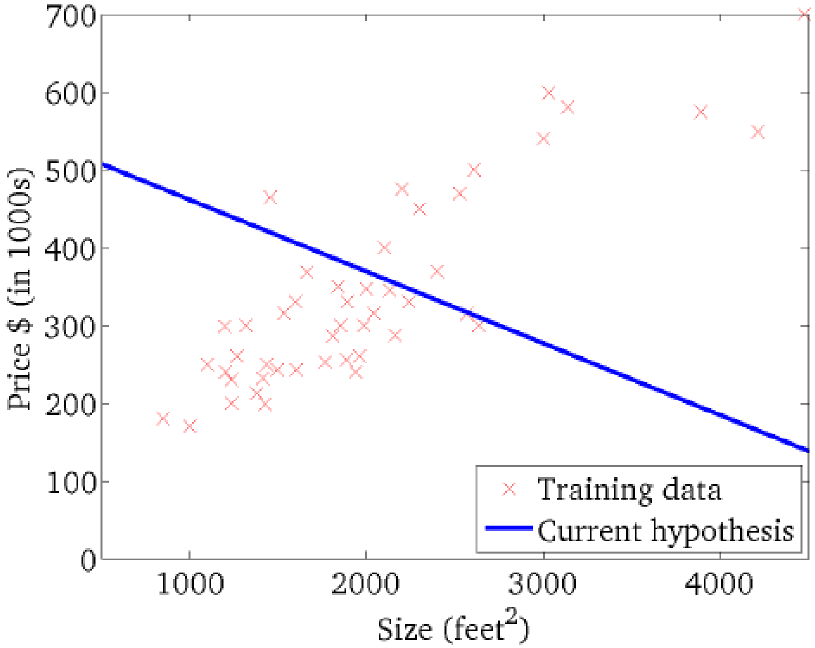
(function of the parameters θ_0, θ_1)



GD Example

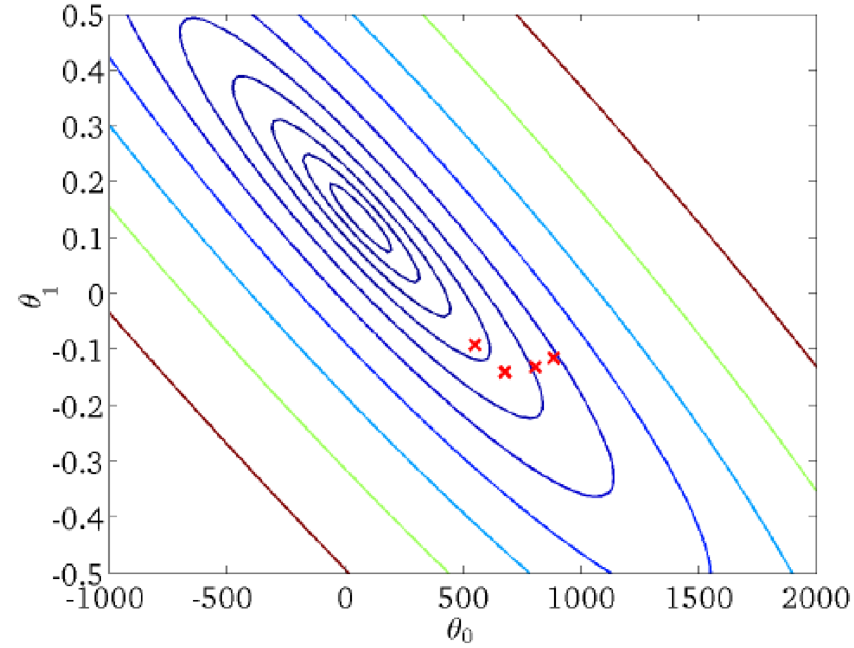
$$h_\theta(x)$$

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



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

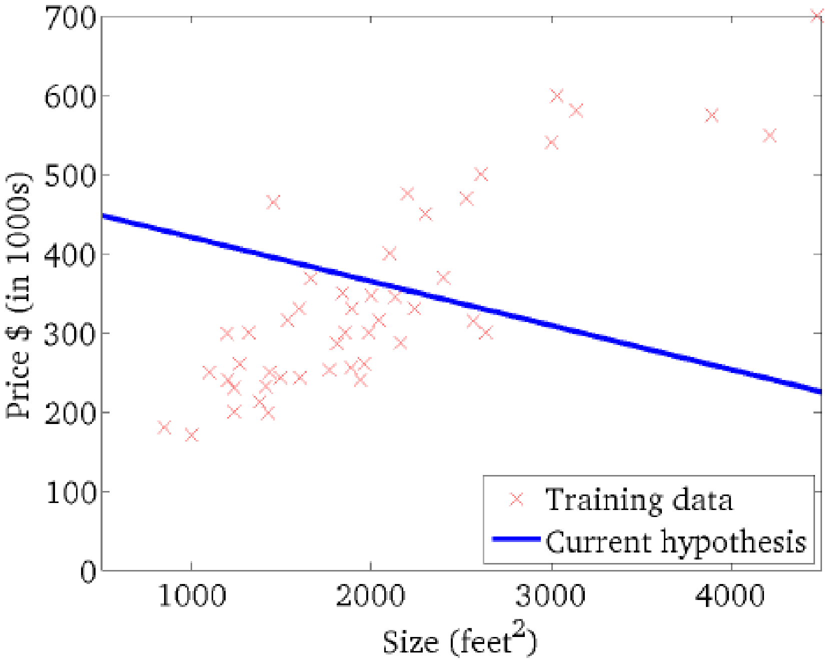
(function of the parameters θ_0, θ_1)



GD Example

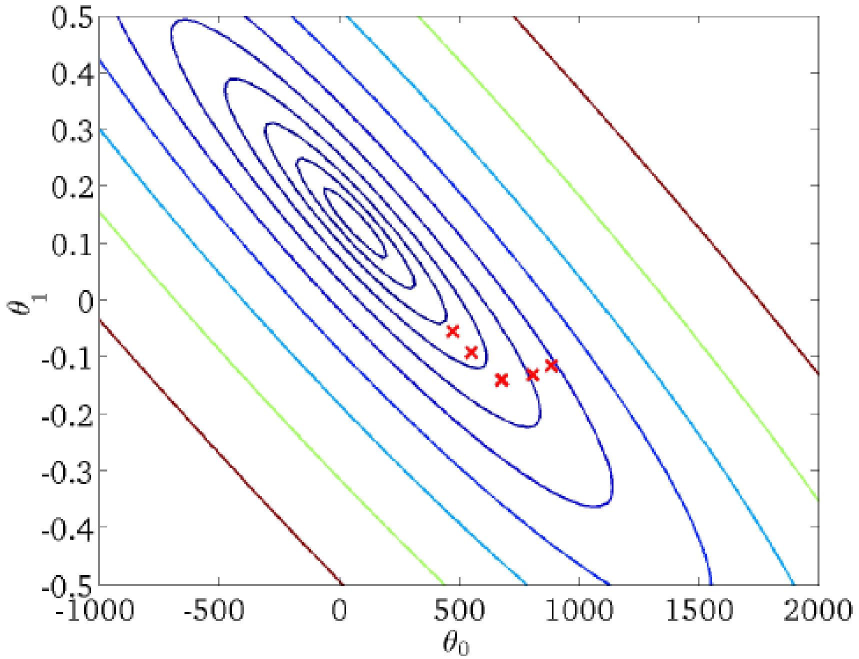
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

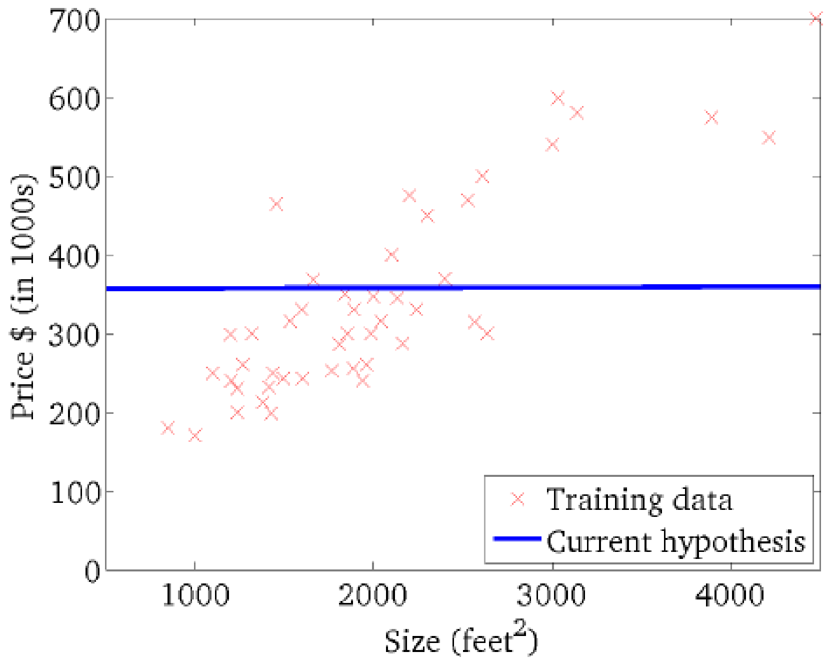
(function of the parameters θ_0, θ_1)



GD Example

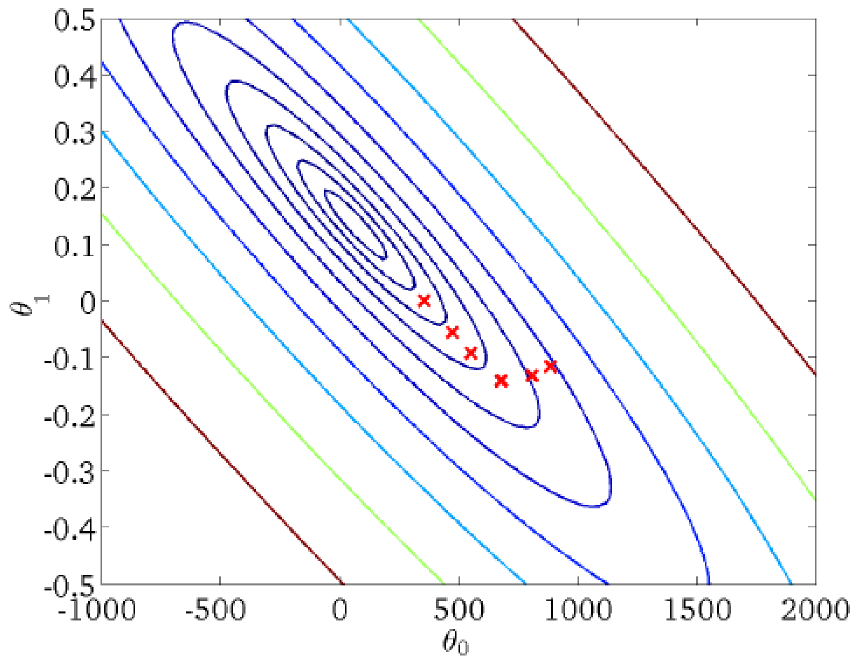
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

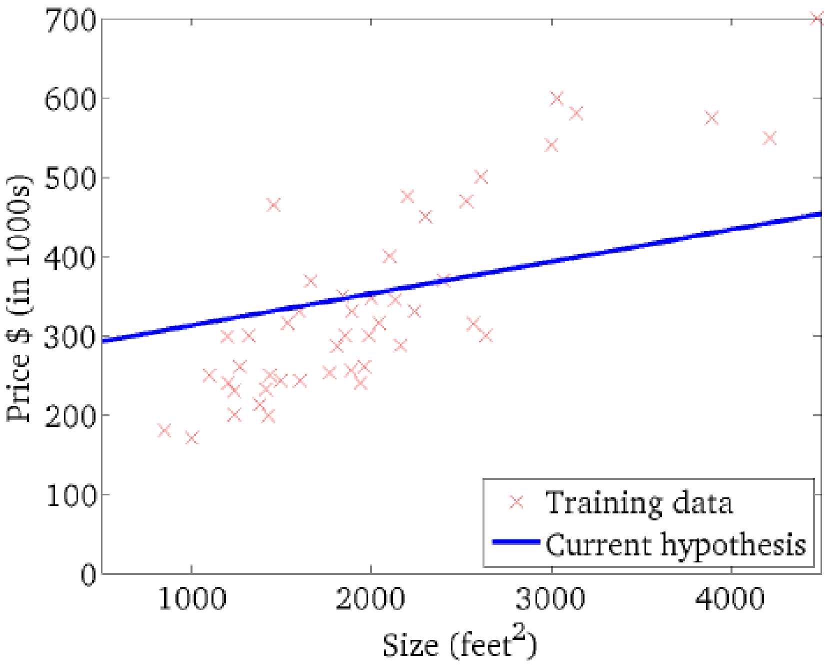
(function of the parameters θ_0, θ_1)



GD Example

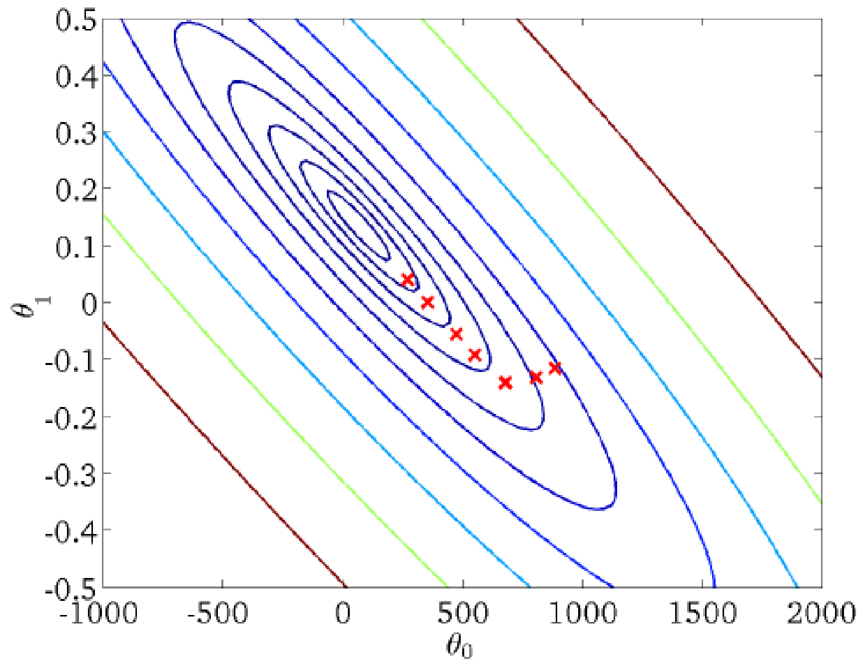
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

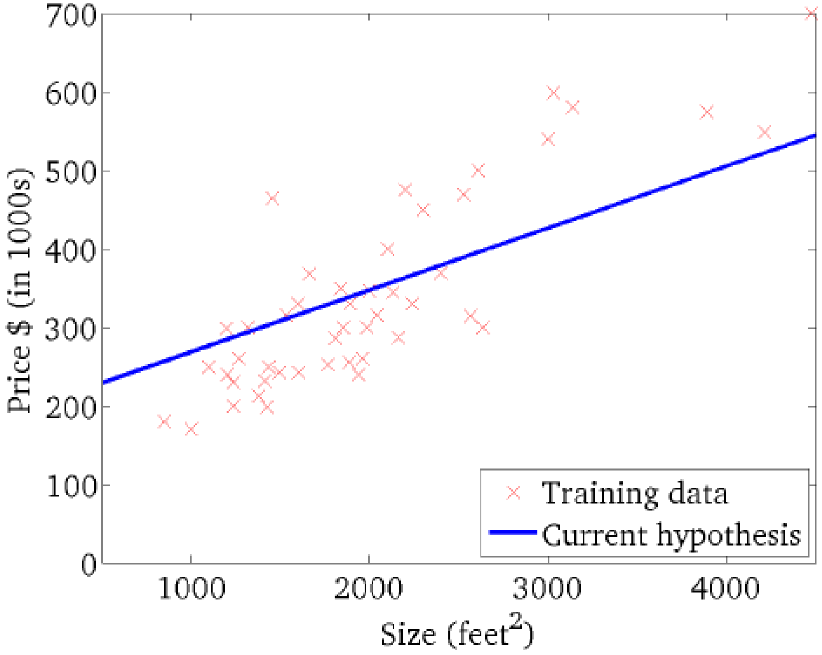
(function of the parameters θ_0, θ_1)



GD Example

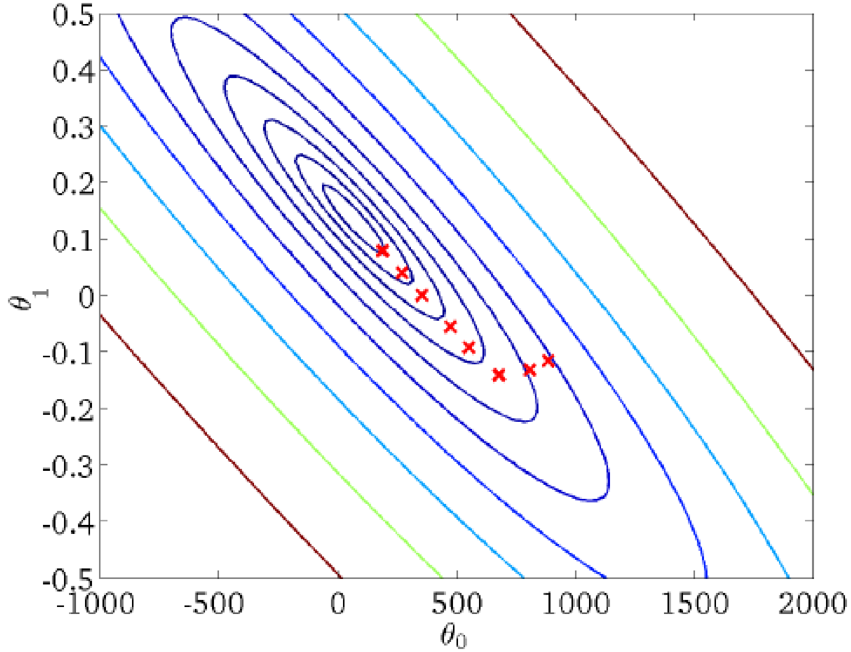
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

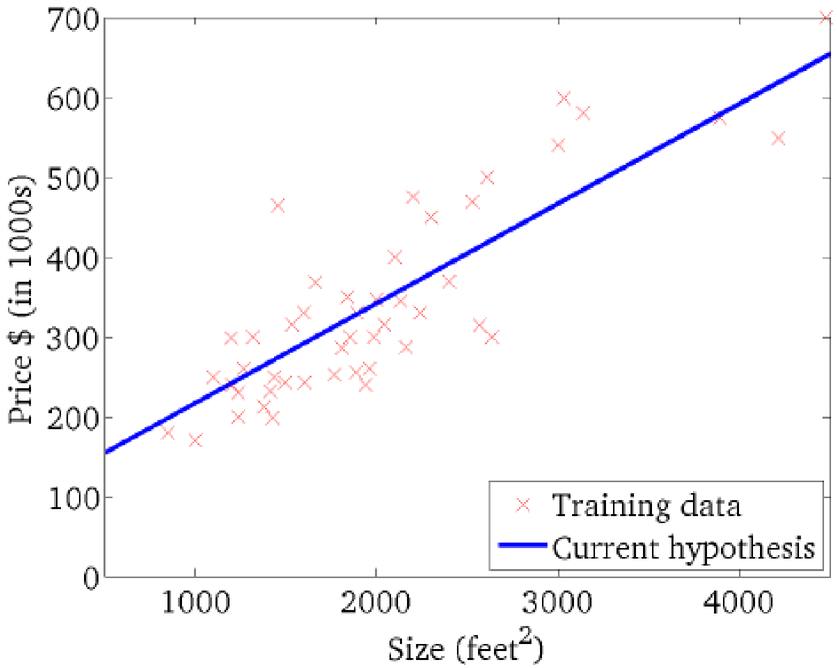
(function of the parameters θ_0, θ_1)



GD Example

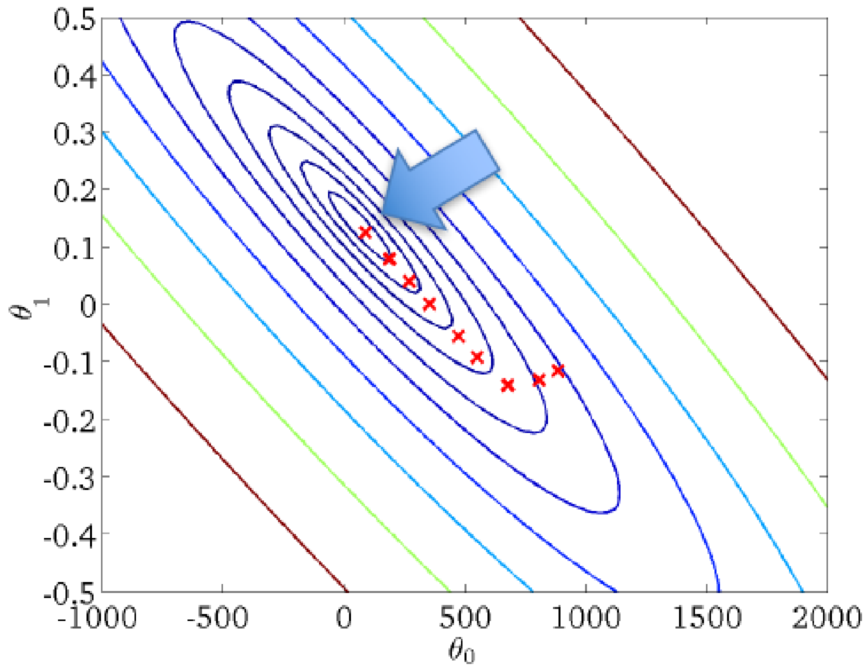
$h_\theta(x)$

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



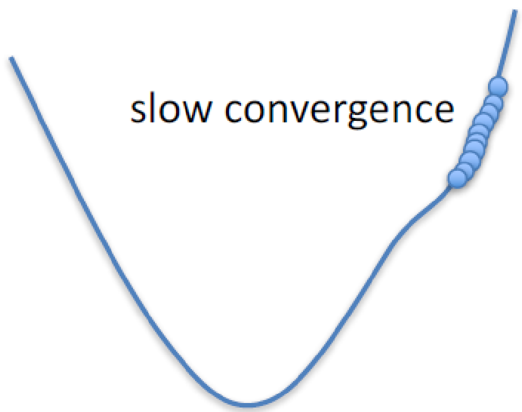
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

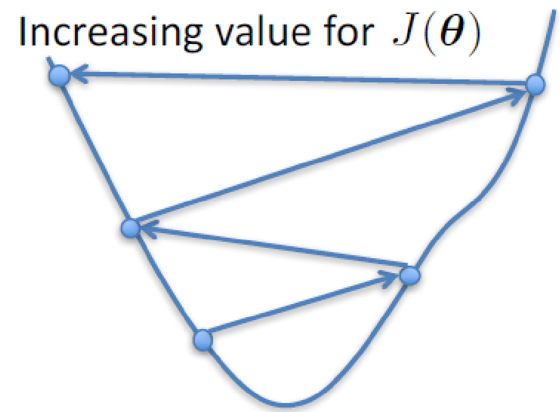


Choosing learning rate

α too small



α too large



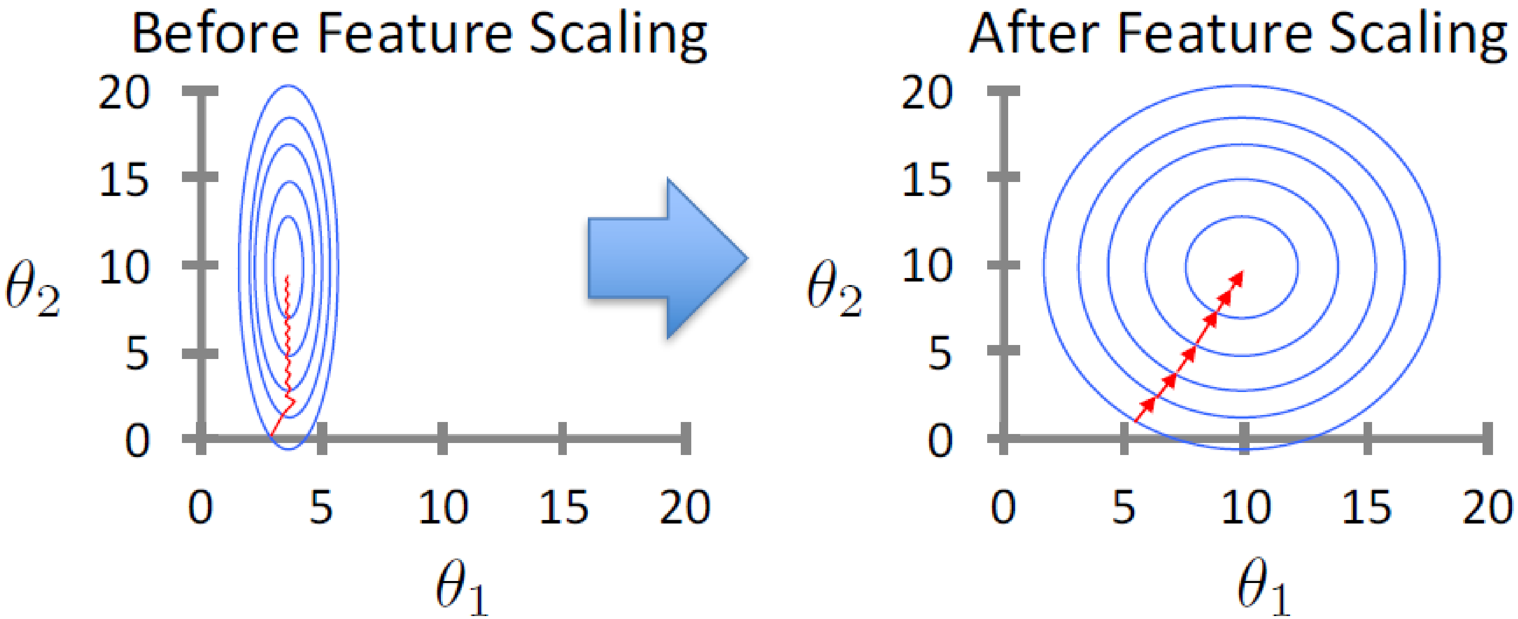
- May overshoot the minimum
- May fail to converge
- May even diverge

To see if gradient descent is working, print out $J(\boldsymbol{\theta})$ each iteration

- The value should decrease at each iteration
- If it doesn't, adjust α

Feature Scaling

- **Idea:** Ensure that feature have similar scales



- Makes gradient descent converge *much* faster

Issues with Gradient Descent

- Might get stuck in local optimum and not converge to global optimum
  - Restart from multiple initial points
- Only works with differentiable loss functions
- Small or large gradients
  - Feature scaling helps
- Tune learning rate
  - Can use line search for determining optimal learning rate

Review

- In practice several techniques can help generate more robust models
  - Outlier removal
  - Feature scaling
- Gradient descent is an efficient algorithm for optimization and training LR
  - The most widely used algorithm in ML!
  - Much faster than using closed-form solution
  - Main issues with Gradient Descent is convergence and getting stuck in local optima

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
- Thanks!