DS 4400

Machine Learning and Data Mining I

Alina Oprea
Associate Professor, CCIS
Northeastern University

September 20 2018

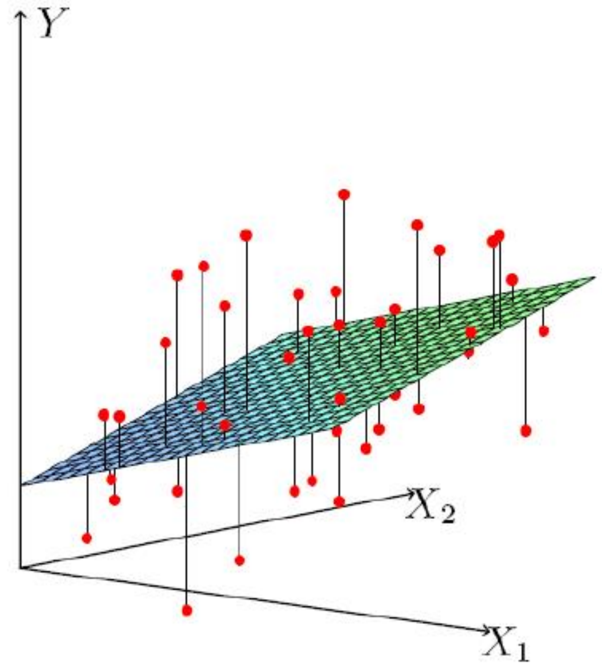
Review

- Solution for multiple linear regression can be computed in closed form
  - Matrix inversion is computationally intense
- 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!

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

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



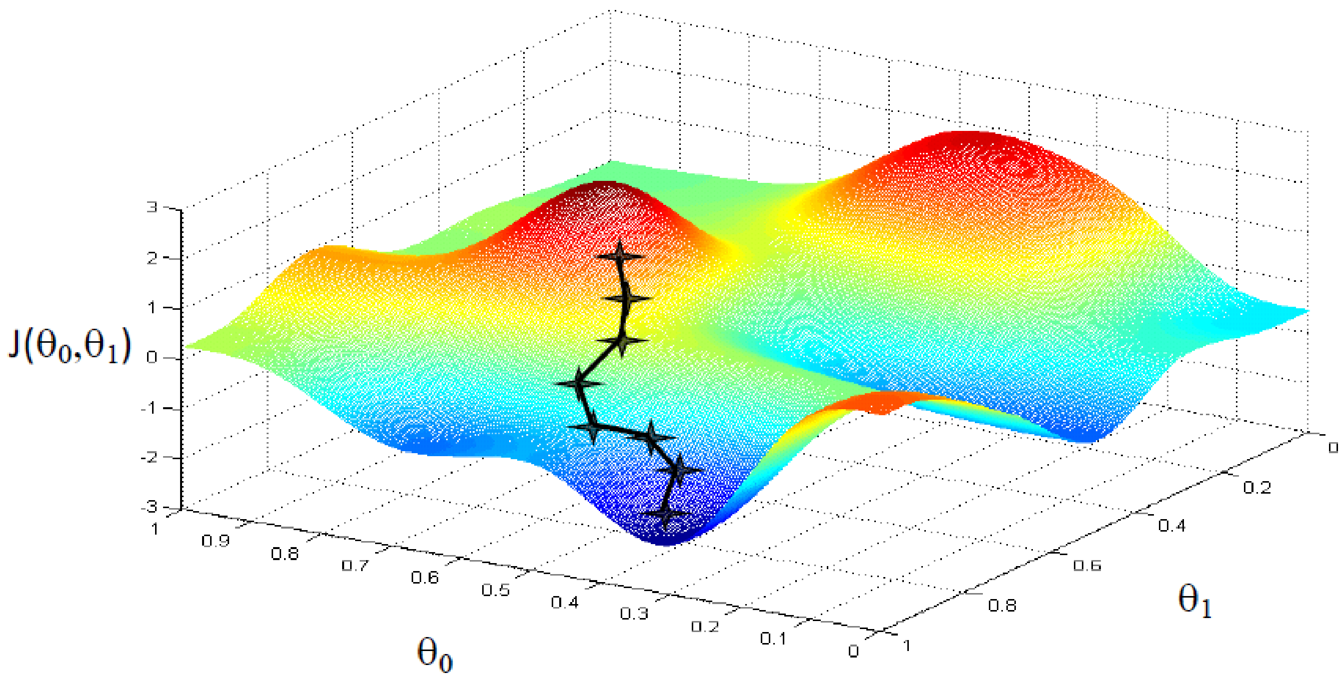
Outline

- Gradient Descent
  - Derivation for simple and multiple Linear Regression
  - Issues with Gradient Descent
  - Comparison with closed-form solution
- Regularization
  - Ridge and Lasso regression
  - Lab example

How to optimize $J(\theta)$?

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

Move in the direction of steepest descent



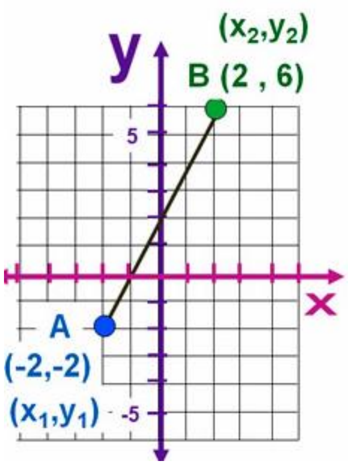
Gradient Descent

- 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

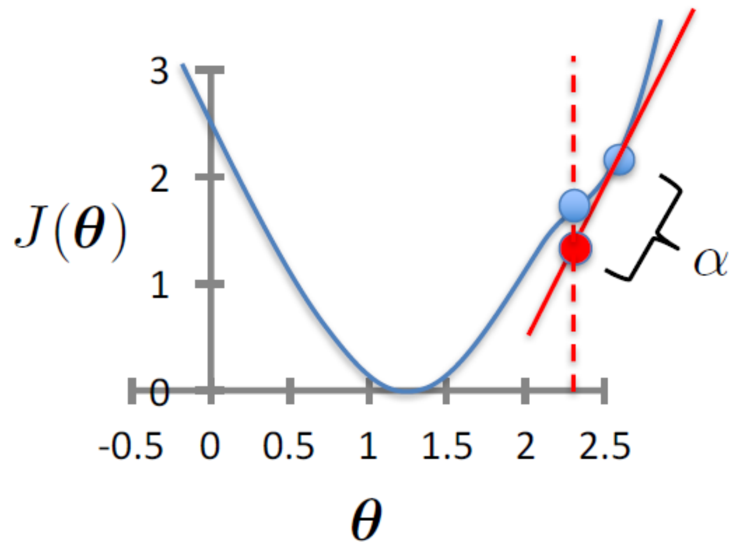


The Gradient "m" is:

$$\boxed{m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta Y}{\Delta X}}$$

$$m = \frac{6 - {}^{-}2}{2 - {}^{-}2}$$

$$m = 8 / 4 = 2 \checkmark$$



Gradient = slope of line tangent
to curve at the same point

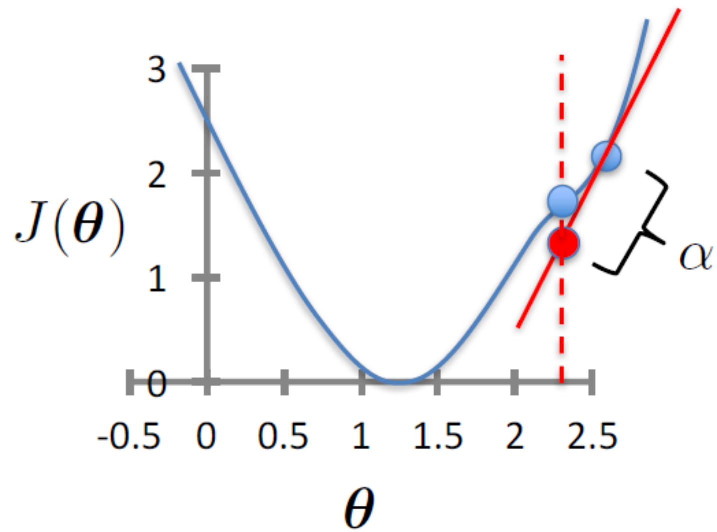
Gradient Descent

- 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



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

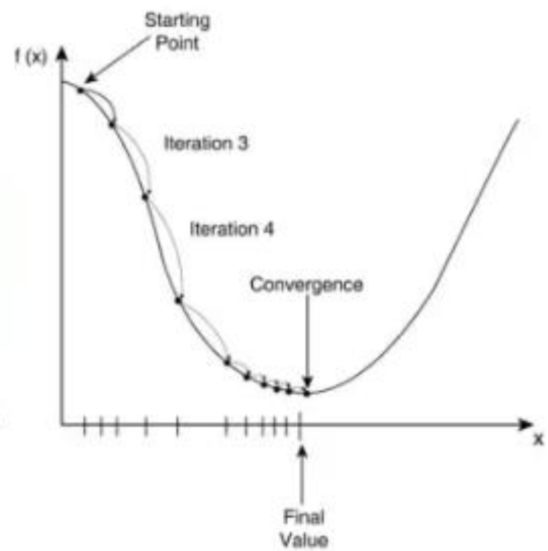
Gradient Descent

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

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

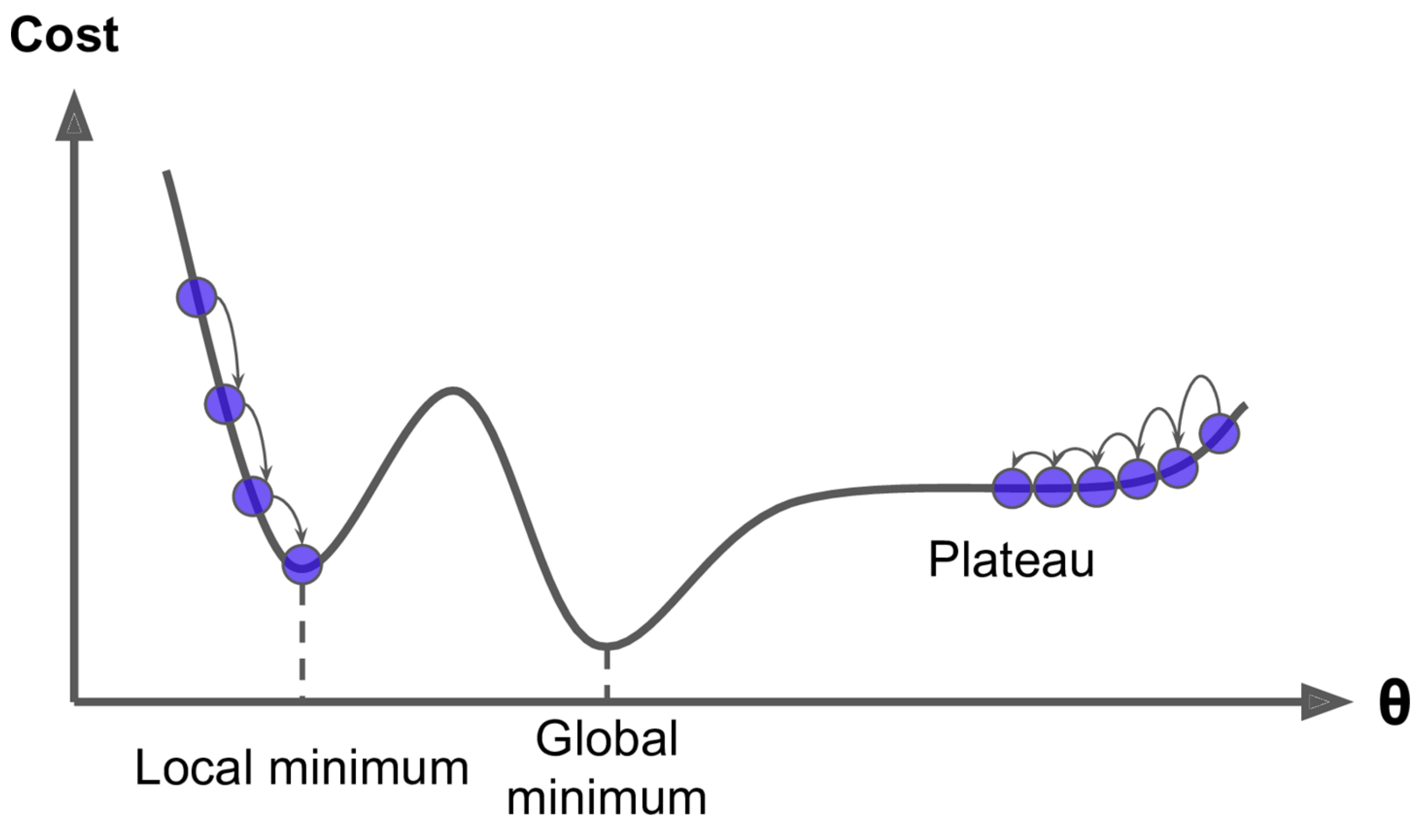
simultaneous update
for j = 0 ... d

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)!

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

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

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 $\|\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old}\|_2 < \epsilon$

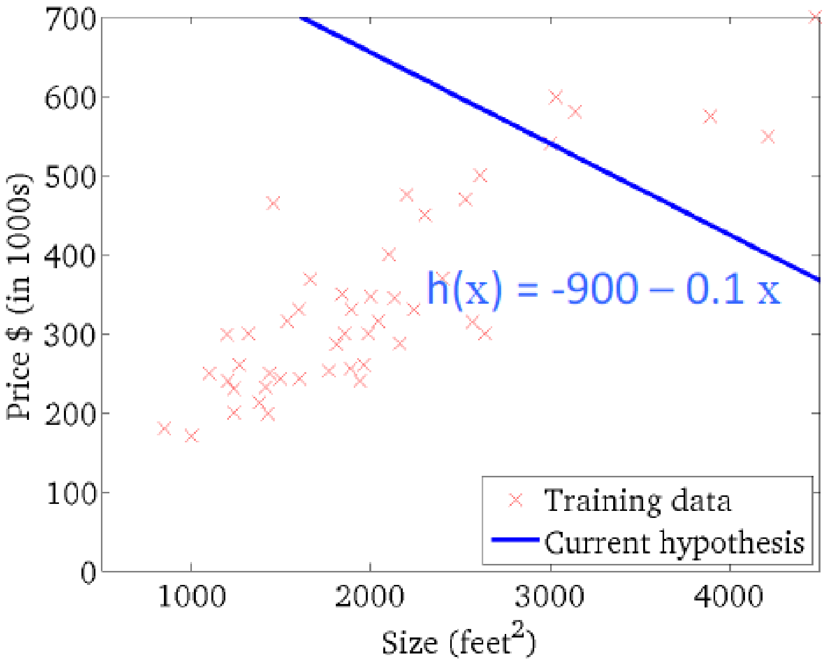
L_2 norm: $$\|\boldsymbol{v}\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \ldots + v_{|v|}^2}$$

Can also bound number of iterations

GD Example

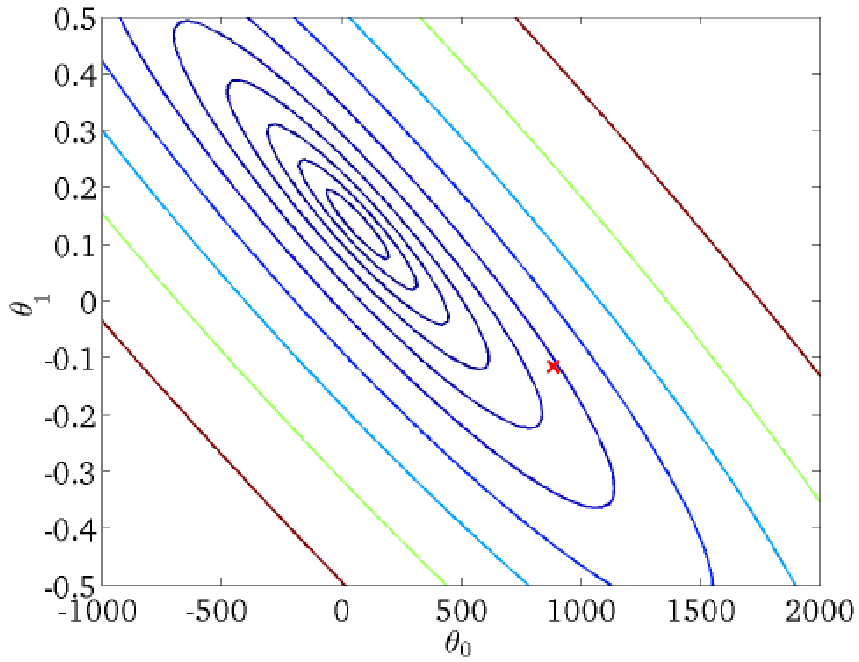
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

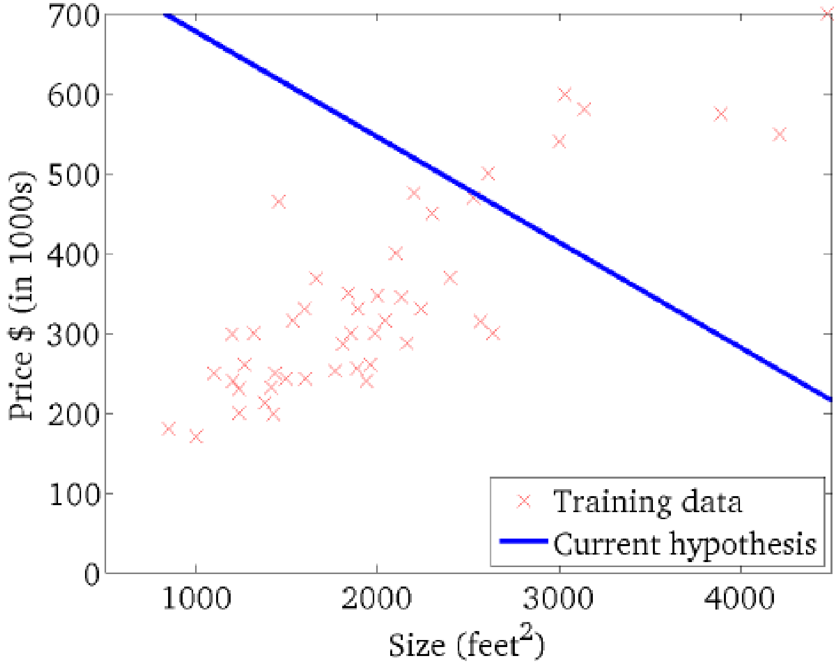
(function of the parameters θ_0, θ_1)



GD Example

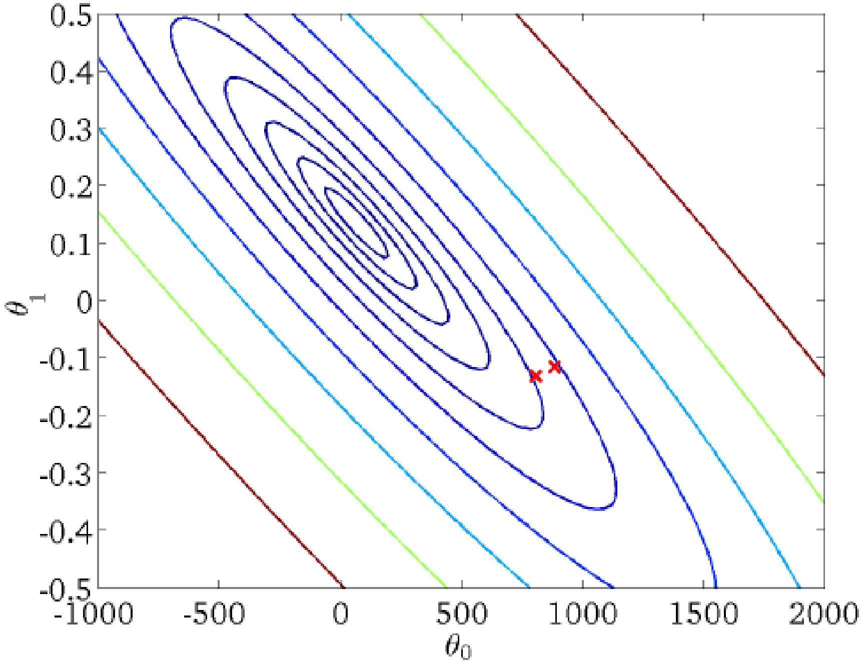
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

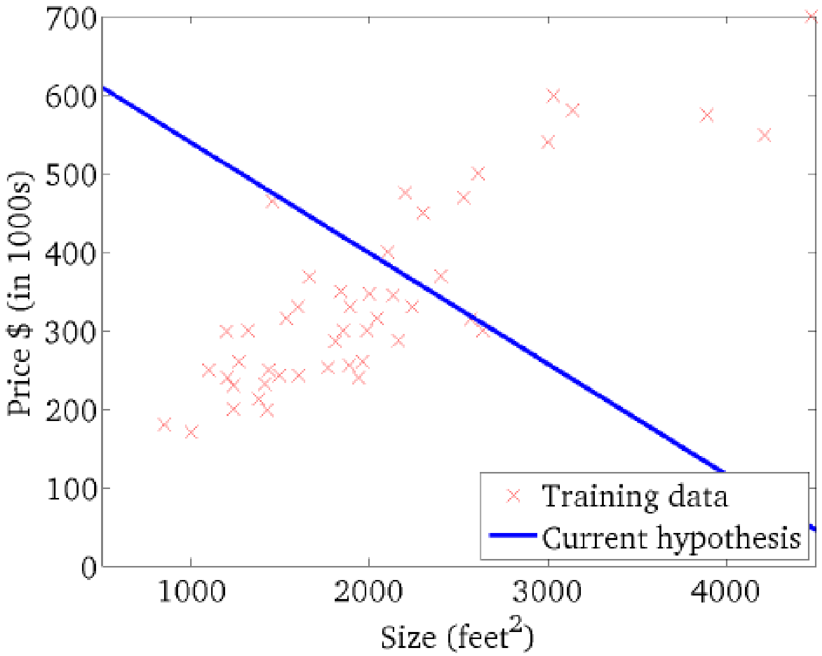
(function of the parameters θ_0, θ_1)



GD Example

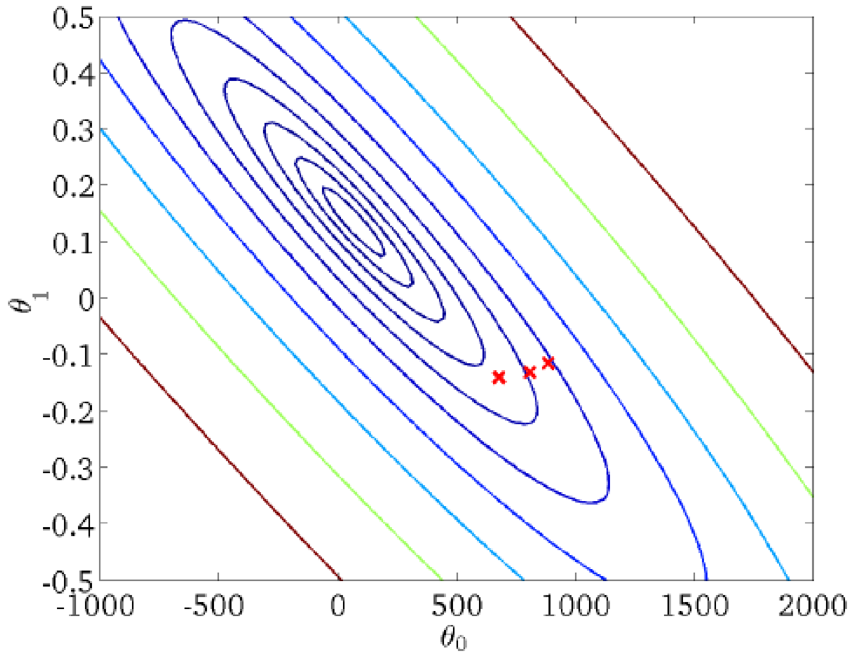
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

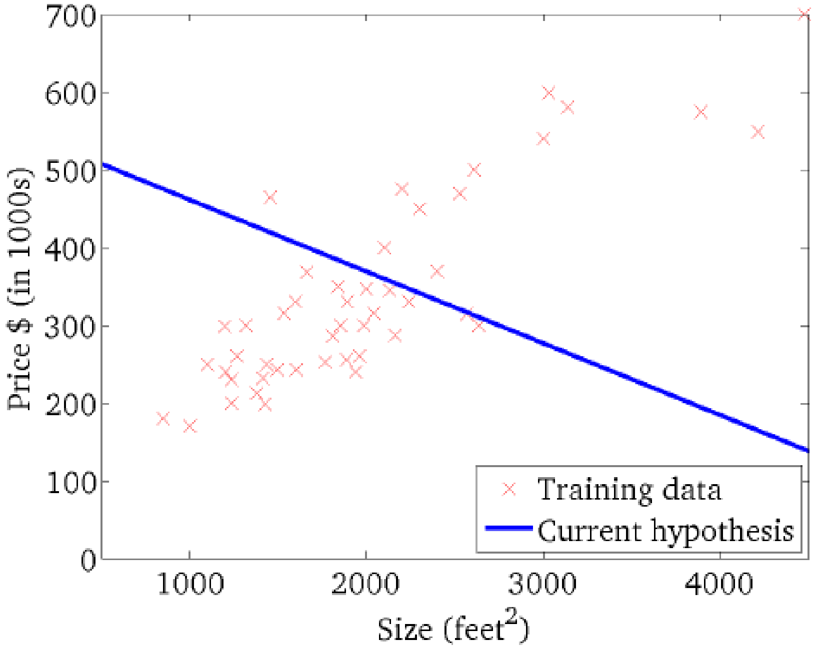
(function of the parameters θ_0, θ_1)



GD Example

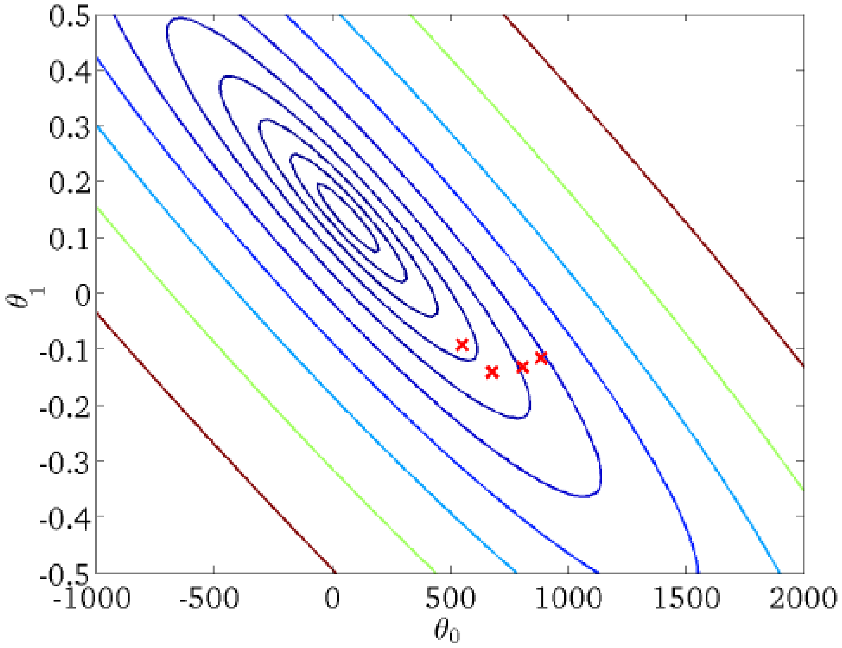
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

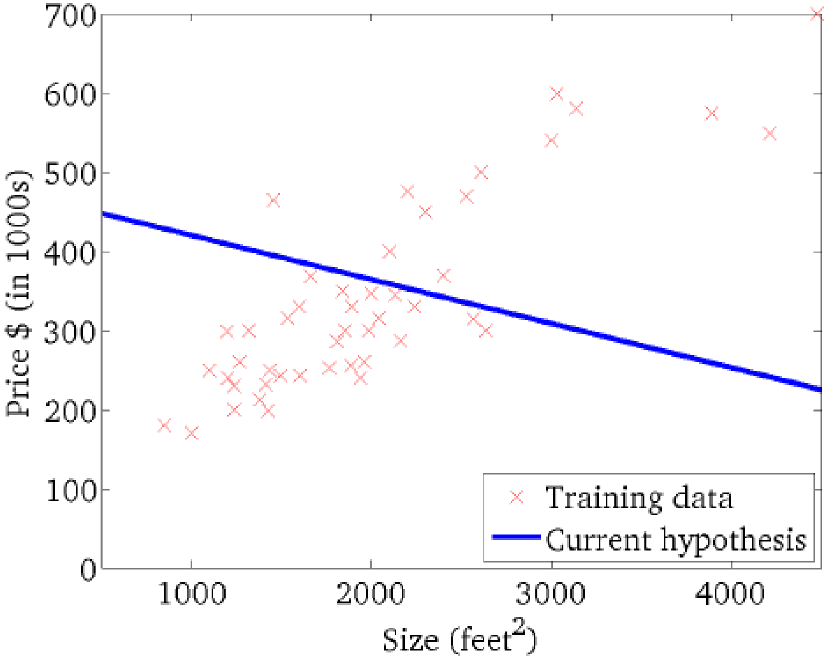
(function of the parameters θ_0, θ_1)



GD Example

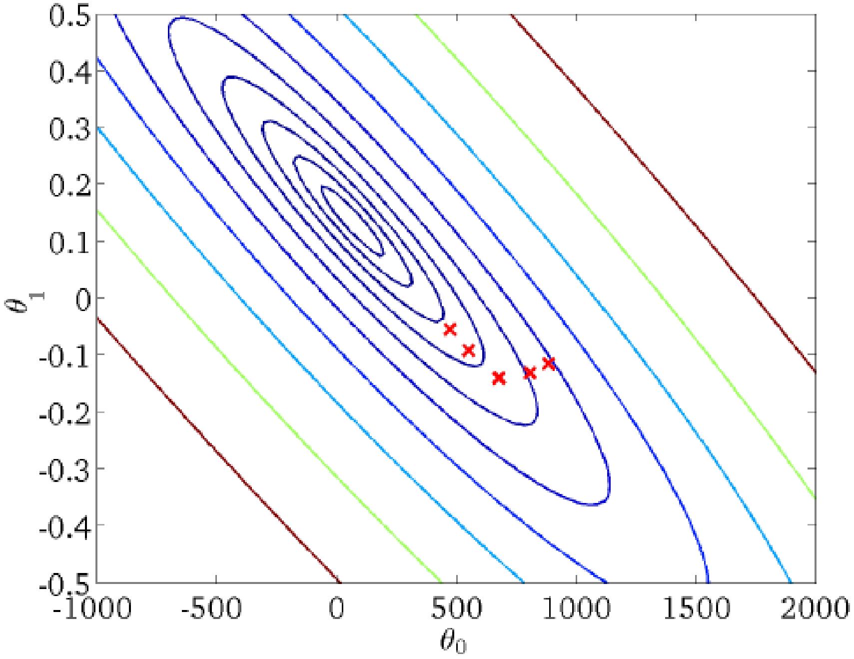
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

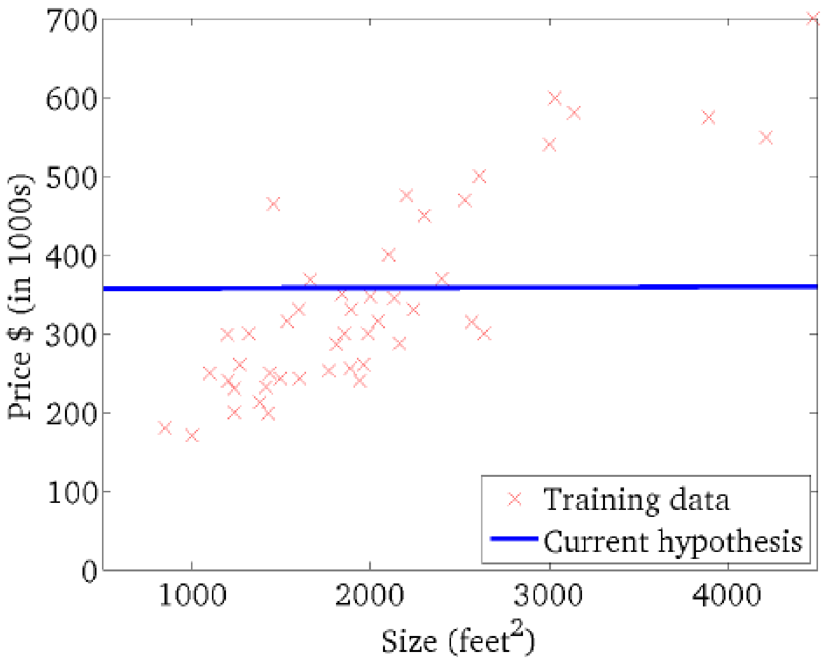
(function of the parameters θ_0, θ_1)



GD Example

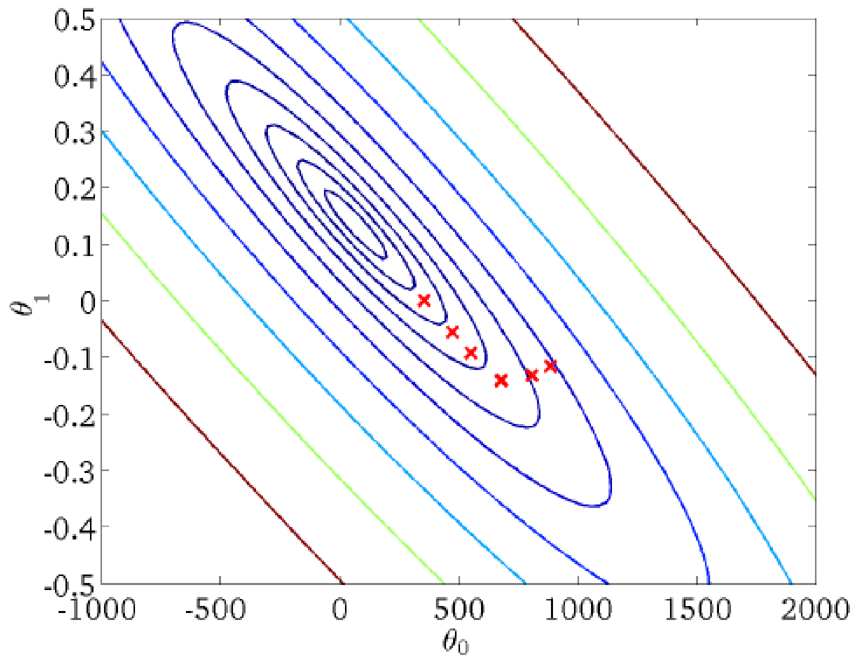
$$h_\theta(x)$$

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



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

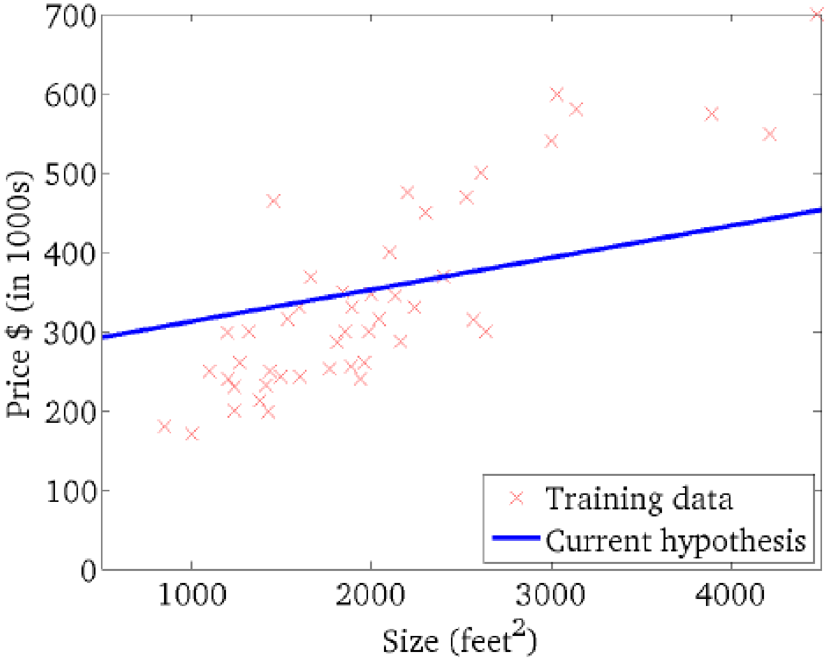
(function of the parameters θ_0, θ_1)



GD Example

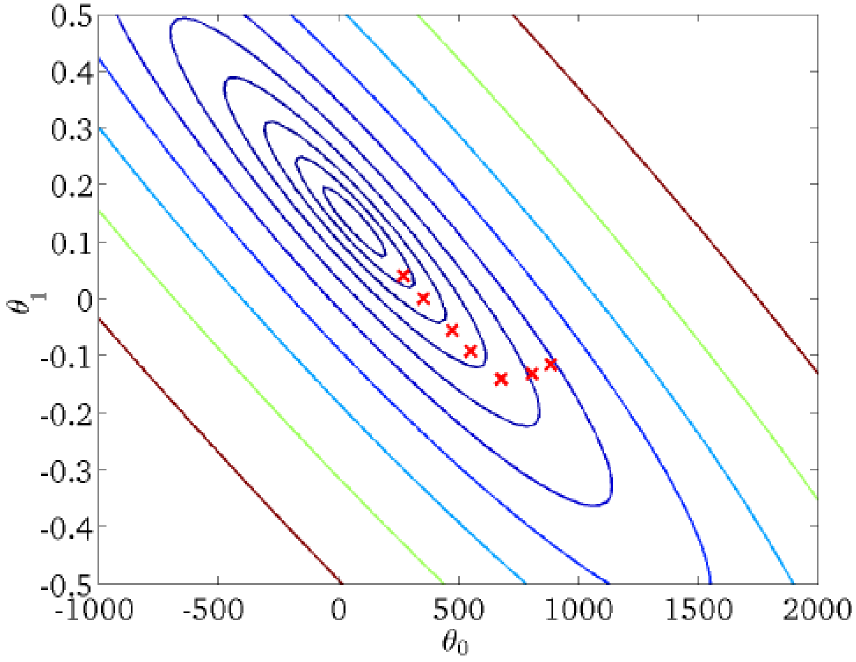
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

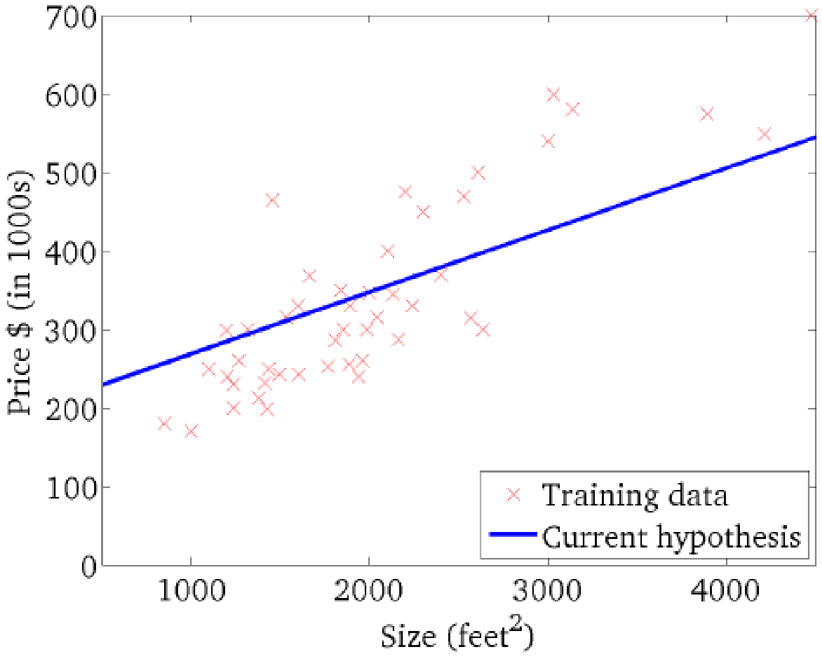
(function of the parameters θ_0, θ_1)



GD Example

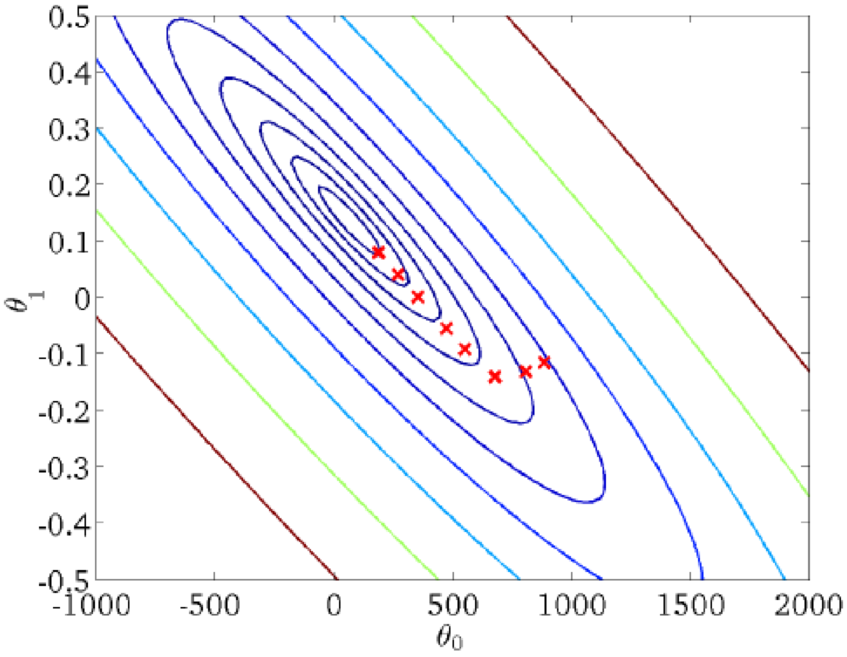
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

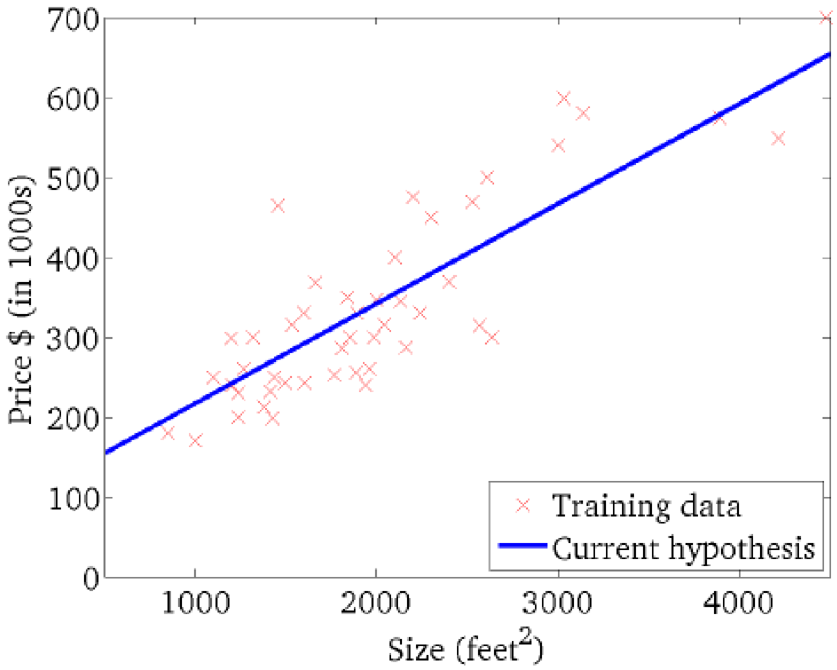
(function of the parameters θ_0, θ_1)



GD Example

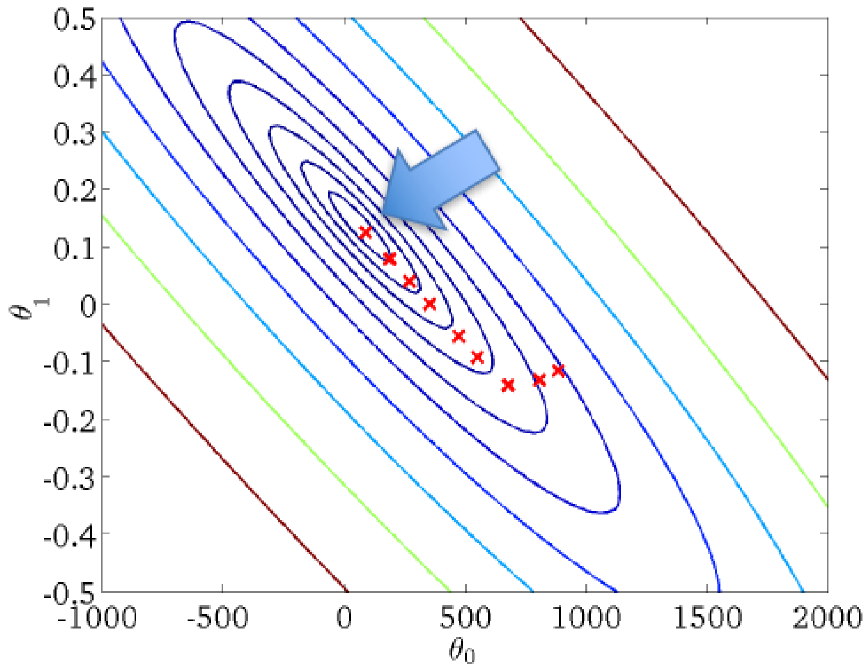
$$h_\theta(x)$$

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



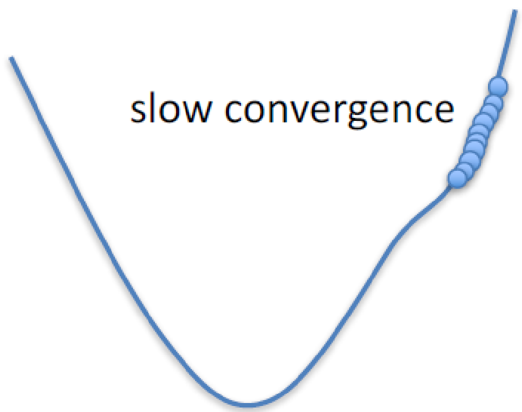
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

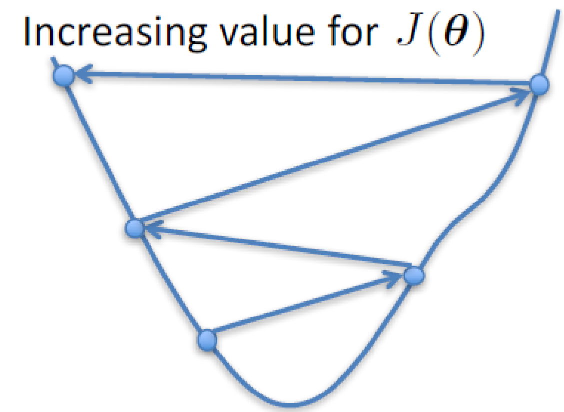


Choosing learning rate

α too small



α too large

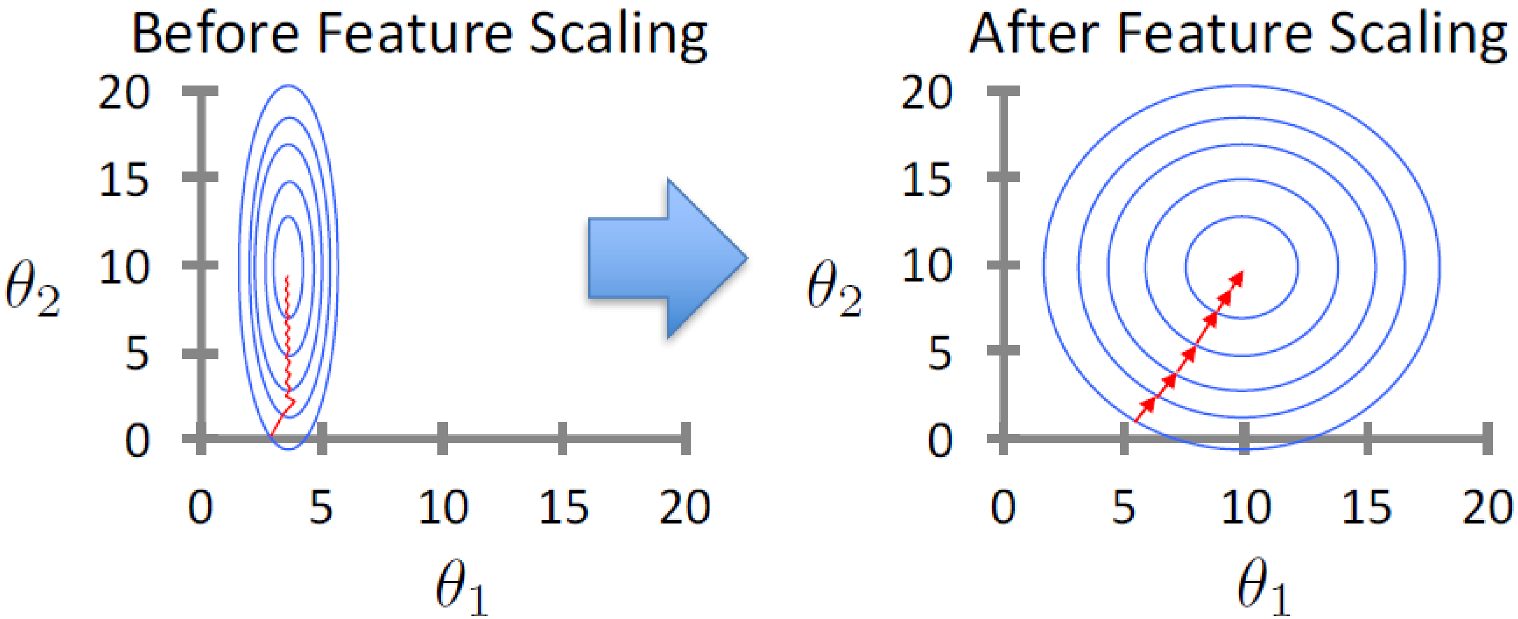


- May overshoot the minimum
- May fail to converge
- May even diverge

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

Gradient Descent vs Closed Form

Gradient Descent

- Initialize θ
- Repeat until convergence

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

simultaneous update for j = 0 ... d

Closed form

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

Gradient Descent	**Closed Form Solution**
• Requires multiple iterations • Need to choose α • Works well when n is large • Can support incremental learning	• Non-iterative • No need for α • Slow if n is large – Computing $(X^\intercal X)^{-1}$ is roughly $O(n^3)$

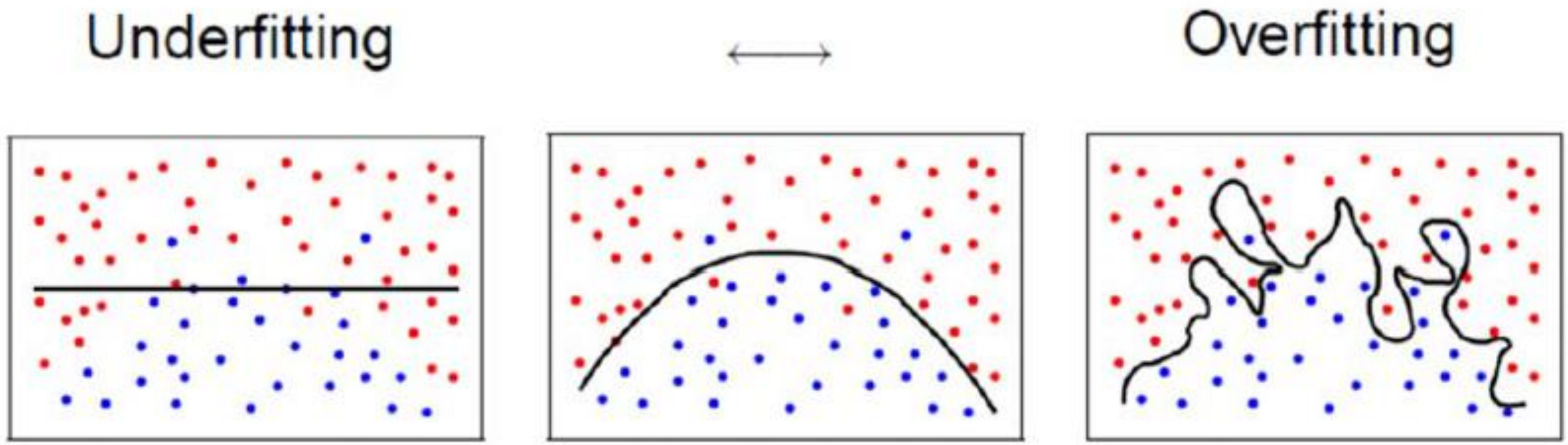
Issues with Gradient Descent

- 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

Outline

- Gradient Descent
  – Derivation for simple and multiple Linear Regression
  – Issues with Gradient Descent
  – Comparison with closed-form solution
- Regularization
  – Ridge and Lasso regression
  – Lab example

Generalization in ML



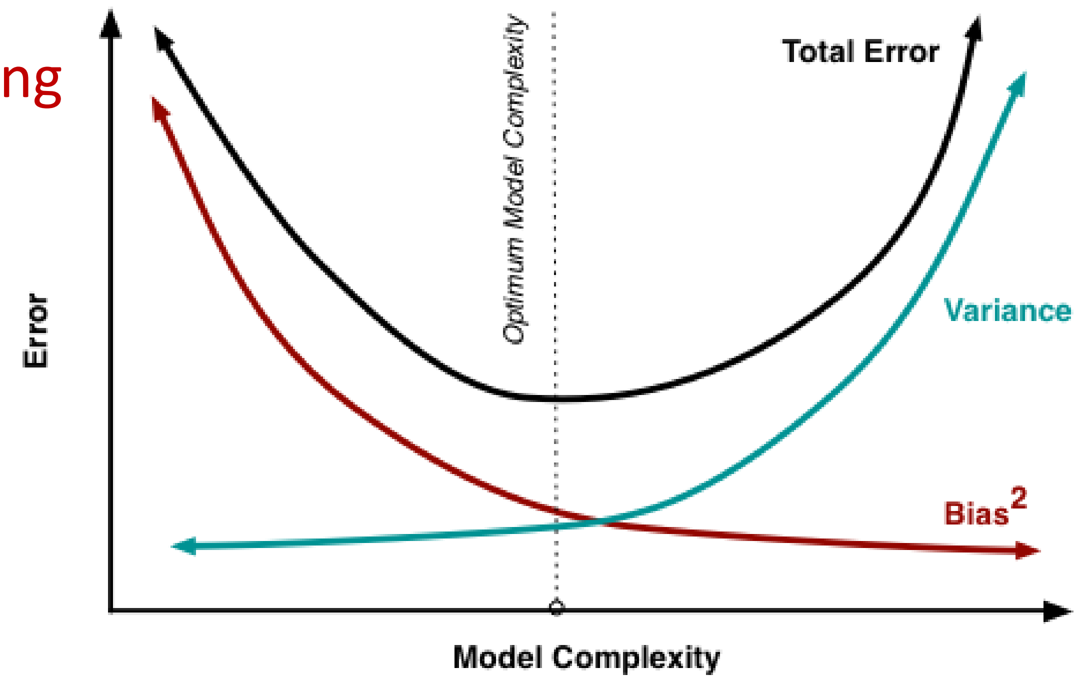
Simple model

Complex model

- Goal is to generalize well on new testing data
- Risk of overfitting to training data
  - MSE close to 0, but performs poorly on test data

Bias-Variance Tradeoff

Under-fitting



Over-fitting

- Bias = Difference between estimated and true models
- Variance = Model difference on different training sets

MSE is proportional to Bias + Variance

Regularization

- A method for automatically controlling the complexity of the learned hypothesis
- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label
- Can also address overfitting by eliminating features (either manually or via model selection)

Reduce model complexity

Reduce model variance

Ridge regression

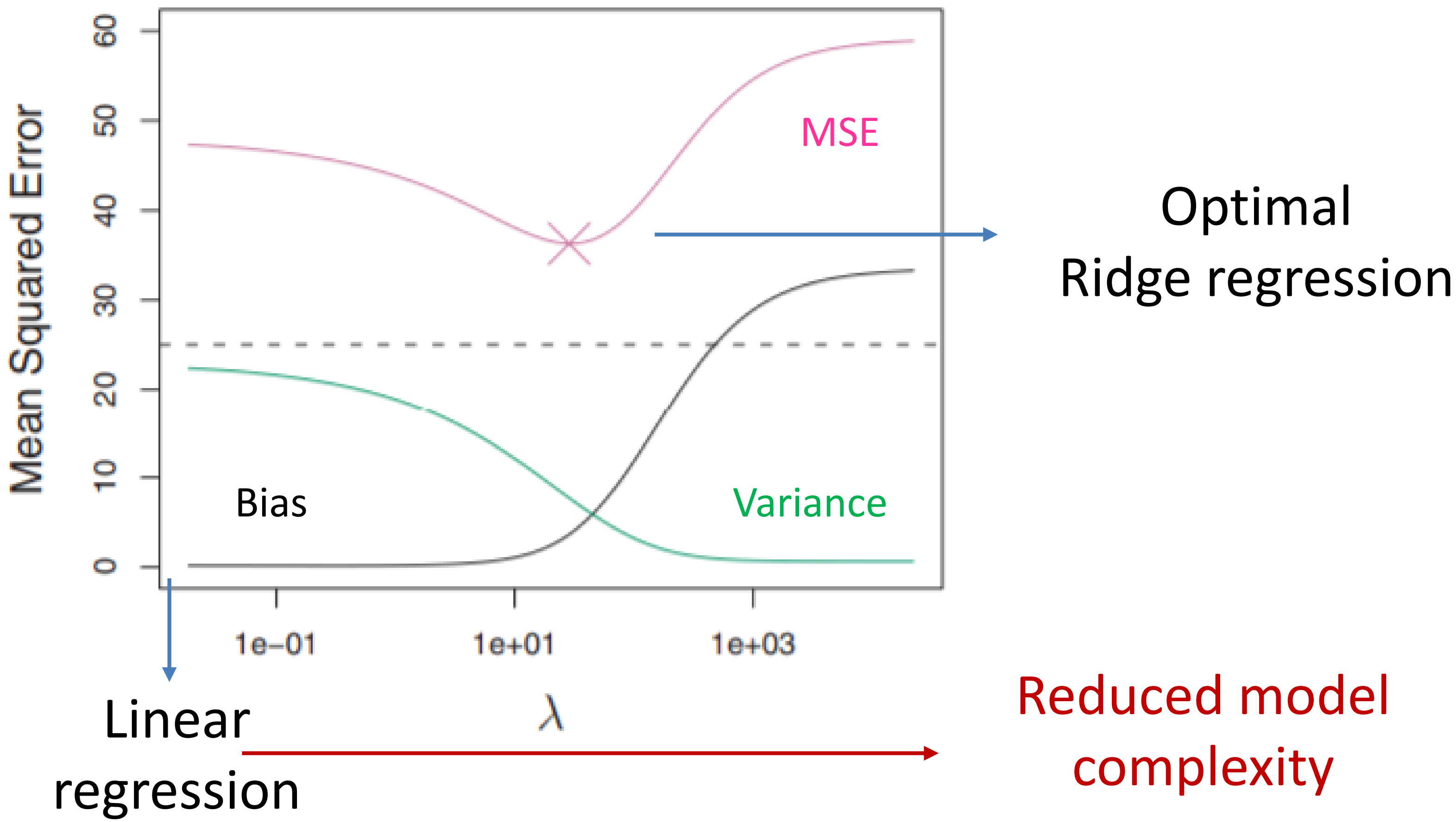
- Linear regression objective function

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!

    - If λ = 0, we train linear regression
    - If λ is large, the coefficients will shrink close to 0

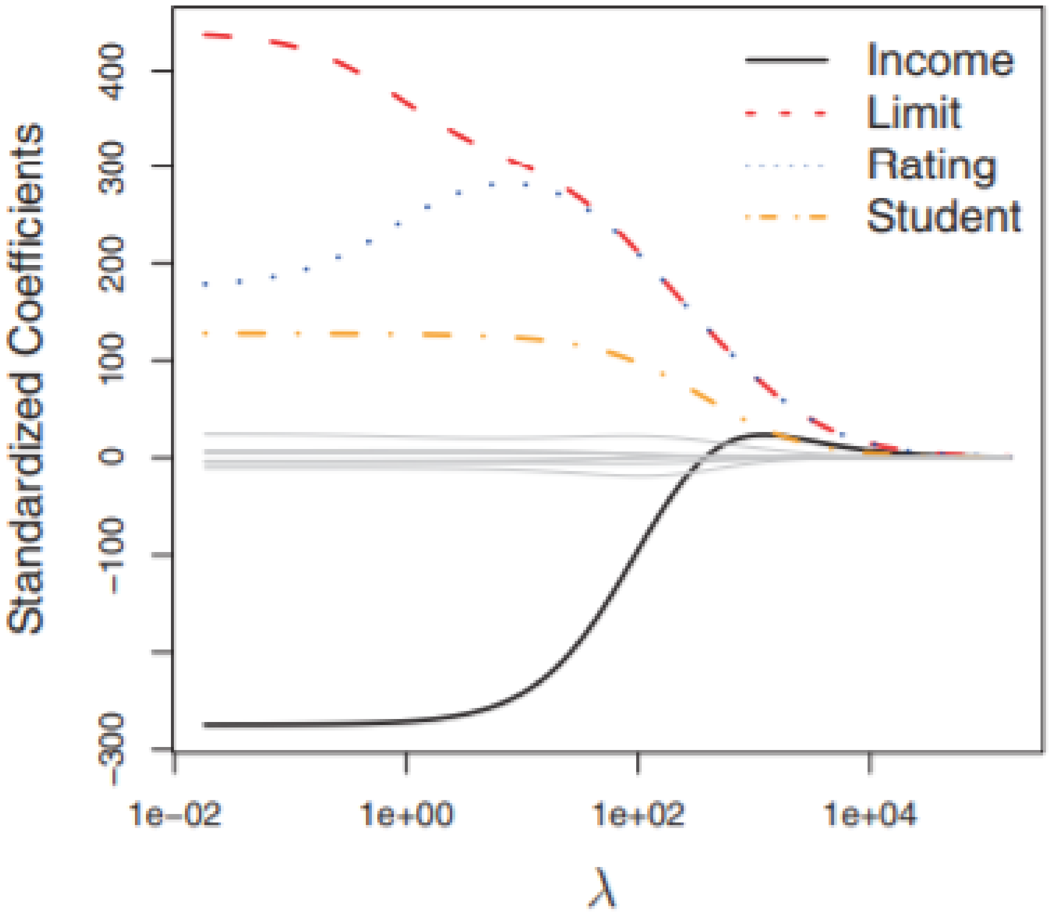
Bias-Variance Tradeoff



Ridge performs better when linear regression has high variance

- Example: d (dimension) is close to n (training set size)

Coefficient shrinkage



Predict credit card balance

GD for Ridge Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} \theta_j^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

GD for Ridge Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} \theta_j^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Gradient update:

$$\frac{\partial}{\partial \theta_0} J(\theta) \qquad \theta_0 \leftarrow \theta_0 - \alpha \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) \qquad \theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)} \underbrace{- \alpha \lambda \theta_j}_{\text{regularization}}$$

$$\theta_j \leftarrow \theta_j (1 - \alpha\lambda) - \alpha \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

Lasso regression

$$J(\theta) = \underbrace{\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}_{\text{Squared Residuals}} + \lambda \underbrace{\sum_{j=1}^{d} |\theta_j|}_{\text{Regularization}}$$

- L1 norm for regularization
- No closed form solution
- Algorithms based on quadratic programming or other optimization techniques

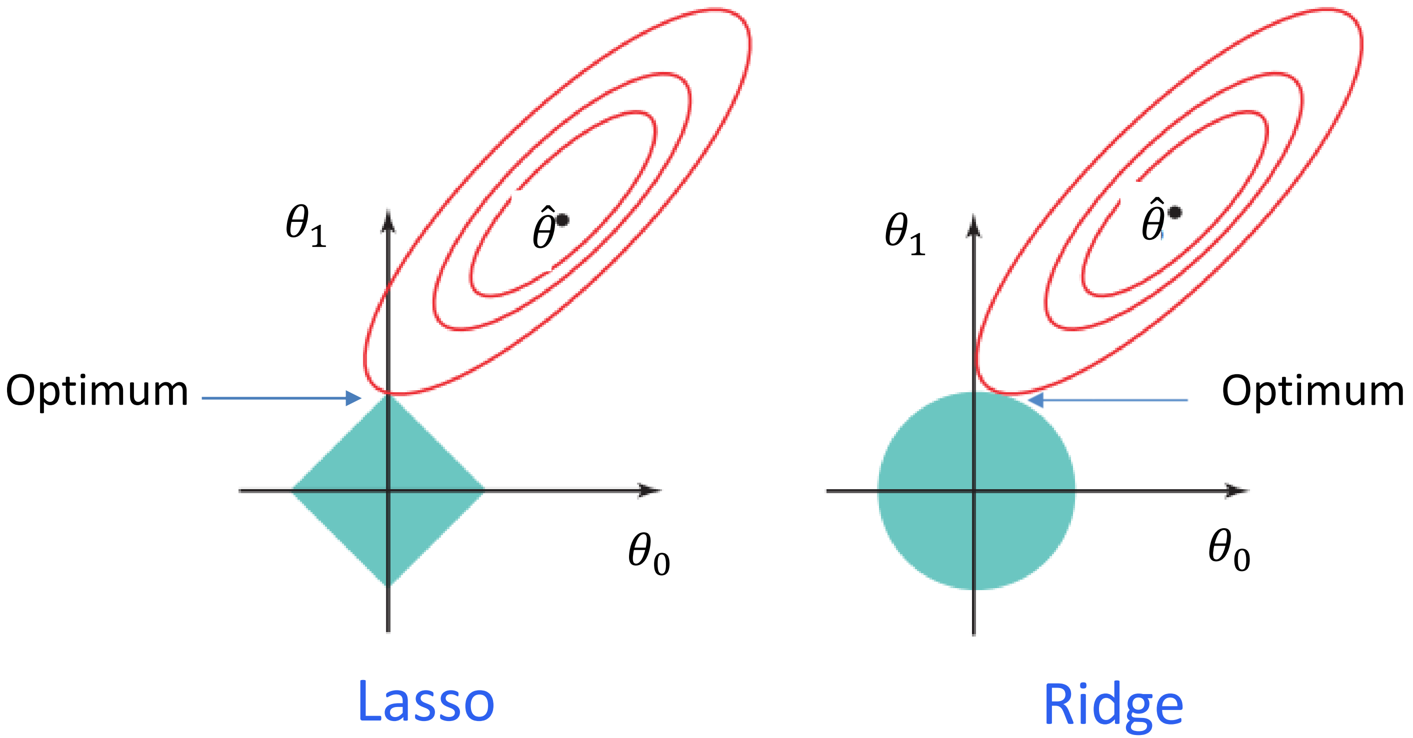
Alternative Formulations

- Ridge
  - L2 Regularization
  - $\min_{\theta} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$ subject to $\sum_{j=1}^{d} \left|\theta_j\right|^2 \leq \epsilon$

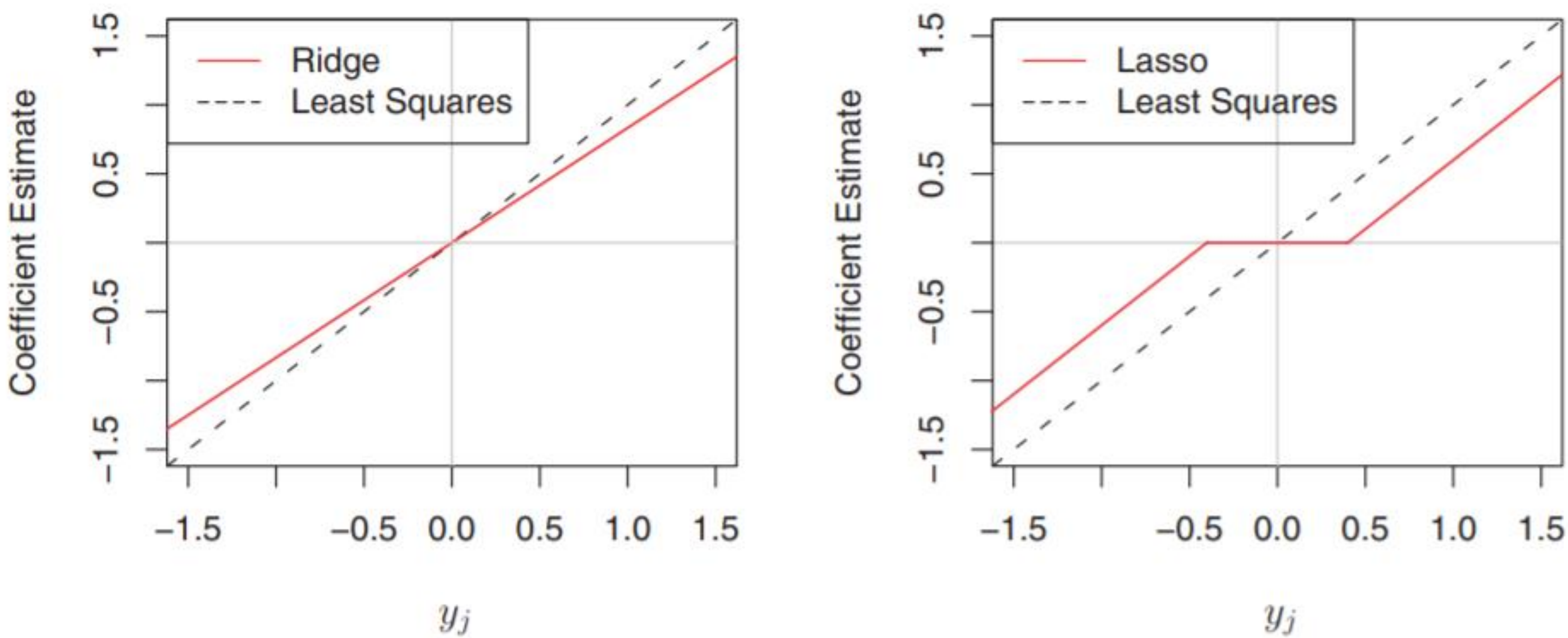
- Lasso
  - L1 regularization
  - $\min_{\theta} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$ subject to $\sum_{j=1}^{d} \left|\theta_j\right| \leq \epsilon$

Lasso vs Ridge

- Ridge shrinks all coefficients
- Lasso sets some coefficients at 0 (sparse solution)
  - Perform feature selection



Lasso vs Ridge



Lab example

```
> library(ISLR)
> fix(Hitters)
```

Data Editor

	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary
1	14	3449	835	69	321	414	375	N	W	632	43	10	475
2	3	1624	457	63	224	266	263	A	W	880	82	14	480
3	11	5628	1575	225	828	838	354	N	E	200	11	3	500
4	2	396	101	12	48	46	33	N	E	805	40	4	91.5
5	11	4408	1133	19	501	336	194	A	W	282	421	25	750
6	2	214	42	1	30	9	24	N	E	76	127	7	70
7	3	509	108	0	41	37	12	A	W	121	283	9	100
8	2	341	86	6	32	34	8	N	W	143	290	19	75
9	13	5206	1332	253	784	890	866	A	E	0	0	0	1100
10	10	4631	1300	90	702	504	488	A	E	238	445	22	517.143
11	9	1876	467	15	192	186	161	N	W	304	45	11	512.5
12	4	1512	392	41	205	204	203	N	E	211	11	7	550
13	6	1941	510	4	309	103	207	A	E	121	151	6	700

Ridge regression

```
> Hitters=na.omit(Hitters)
> x=model.matrix(Salary~.,Hitters)[,-1]
> y=Hitters$Salary
> ridge.mod=glmnet(x,y,alpha=0,lambda=5000)
> coef(ridge.mod)
20 x 1 sparse Matrix of class "dgCMatrix"
                       s0
(Intercept) 305.016480230
AtBat         0.065738413
Hits          0.255494042
HmRun         0.902148872
Runs          0.419912564
RBI           0.428768355
Walks         0.533942922
Years         1.892781352
CAtBat        0.005532745
CHits         0.020876841
CHmRun        0.156069996
CRuns         0.041877748
CRBI          0.043262917
CWalks        0.043634641
LeagueN       1.117728148
DivisionW   -13.063063667
PutOuts       0.033021805
Assists       0.004993208
Errors       -0.061932828
NewLeagueN    1.269197088
> sqrt(sum(coef(ridge.mod)[-1]^2))
[1] 13.36602
```

Data processing (omit N/A)

Fit ridge regression

Coefficient values

Coefficient norm

Ridge regression

```
> ridge.mod=glmnet(x,y,alpha=0,lambda=50)
> coef(ridge.mod)
20 x 1 sparse Matrix of class "dgCMatrix"
                        s0
(Intercept)   4.800582e+01
AtBat        -3.532997e-01
Hits          1.950804e+00
HmRun        -1.286413e+00
Runs          1.158693e+00
RBI           8.114814e-01
Walks         2.709241e+00
Years        -6.179435e+00
CAtBat        6.262426e-03
CHits         1.072029e-01
CHmRun        6.284707e-01
CRuns         2.155421e-01
CRBI          2.148524e-01
CWalks       -1.483366e-01
LeagueN       4.585236e+01
DivisionW    -1.182395e+02
PutOuts       2.501361e-01
Assists       1.206414e-01
Errors       -3.277654e+00
NewLeagueN   -9.424451e+00
> sqrt(sum(coef(ridge.mod)[-1]^2))
[1] 127.4217
```

Fit ridge regression

Coefficient values

Coefficient norm

λ controls parameter size

Lasso regression

```
> lasso.mod=glmnet(x,y,alpha=1,lambda=50)
> coef(lasso.mod)
20 x 1 sparse Matrix of class "dgCMatrix"
                     s0
(Intercept)  88.6306382
AtBat         .
Hits          1.5877156
HmRun         .
Runs          .
RBI           .
Walks         1.8197051
Years         .
CAtBat        .
CHits         .
CHmRun        .
CRuns         0.1711419
CRBI          0.3709268
CWalks        .
LeagueN       .
DivisionW   -43.3646551
PutOuts       0.1341253
Assists       .
Errors        .
NewLeagueN    .
> sqrt(sum(coef(lasso.mod)[-1]^2))
[1] 43.43398
```

Fit Lasso regression

13 coefficients set at zero

Coefficient norm

Acknowledgements

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