Linear Regression

CE-717: Machine Learning

Sharif University of Technology

M. Soleymani

Fall 2016

Topics

- Linear regression
  - Error (cost) function
  - Optimization
  - Generalization

Regression problem

- The goal is to make (real valued) predictions given features

- Example: predicting house price from 3 attributes

Size (m^2)	Age (year)	Region	Price (10^6T)
100	2	5	500
80	25	3	250
…	…	…	…

Learning problem

- Selecting a **hypothesis space**
  - Hypothesis space: a set of mappings from feature vector to target

- **Learning (estimation)**: optimization of a cost function
  - Based on the training set $D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ and a cost function we find (an estimate) $f \in F$ of the target function

- **Evaluation**: we measure how well $\hat{f}$ generalizes to unseen examples

Learning problem

- Selecting a **hypothesis space**
  - Hypothesis space: a set of mappings from feature vector to target

- **Learning (estimation)**: optimization of a cost function
  - Based on the training set $D = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$ and a cost function we find (an estimate) $f \in F$ of the target function

- **Evaluation**: we measure how well $\hat{f}$ generalizes to unseen examples

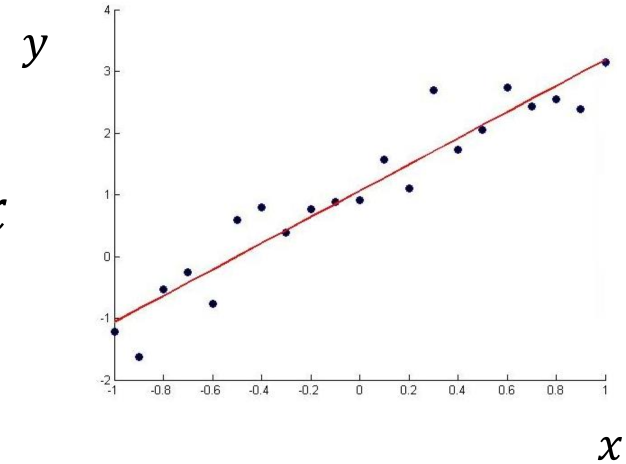
Hypothesis space

- Specify the class of functions (e.g., linear)

- We begin by the class of linear functions
  - easy to extend to generalized linear and so cover more complex regression functions

Linear regression: hypothesis space

- Univariate



$$f : \mathbb{R} \rightarrow \mathbb{R} \quad f(x; \boldsymbol{w}) = w_0 + w_1 x$$

- Multivariate

$$f : \mathbb{R}^d \rightarrow \mathbb{R} \quad f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x_1 + \ldots w_d x_d$$

$\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^T$ are parameters we need to set.

Learning problem

- Selecting a **hypothesis space**
  - Hypothesis space: a set of mappings from feature vector to target

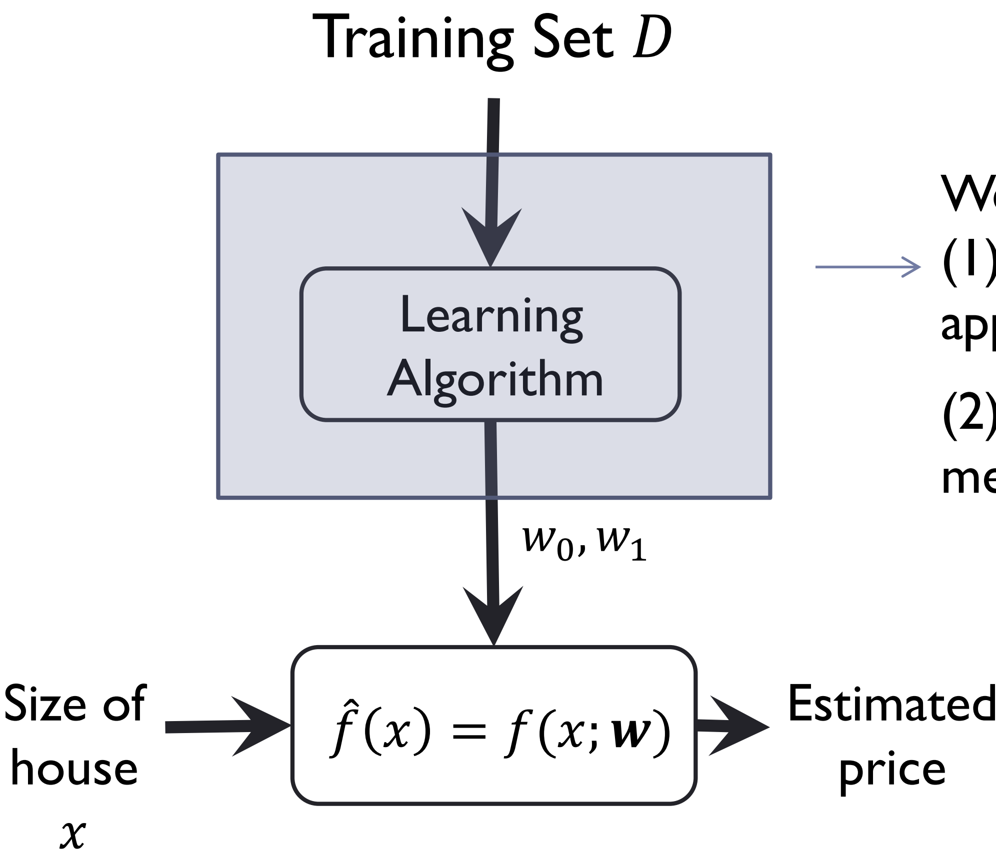
- **Learning (estimation)**: optimization of a cost function
  - Based on the training set $D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ and a cost function we find (an estimate) $f \in F$ of the target function

- **Evaluation**: we measure how well $\hat{f}$ generalizes to unseen examples

Learning algorithm

- Select how to measure the error (i.e. prediction loss)
- Find the minimum of the resulting error or cost function

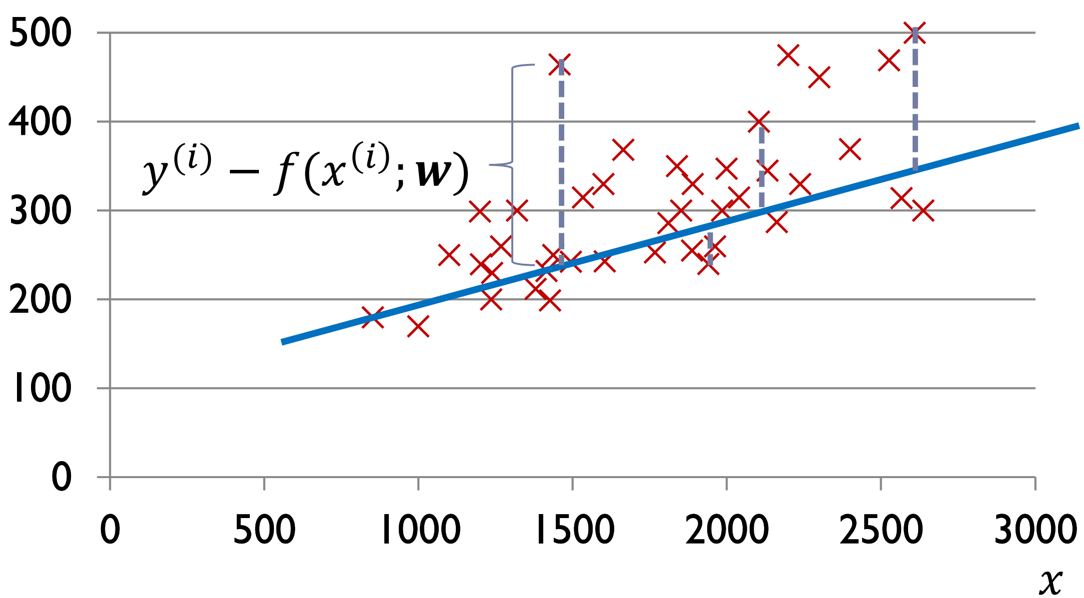
Learning algorithm



We need to
(1) measure how well $f(x; \boldsymbol{w})$ approximates the target

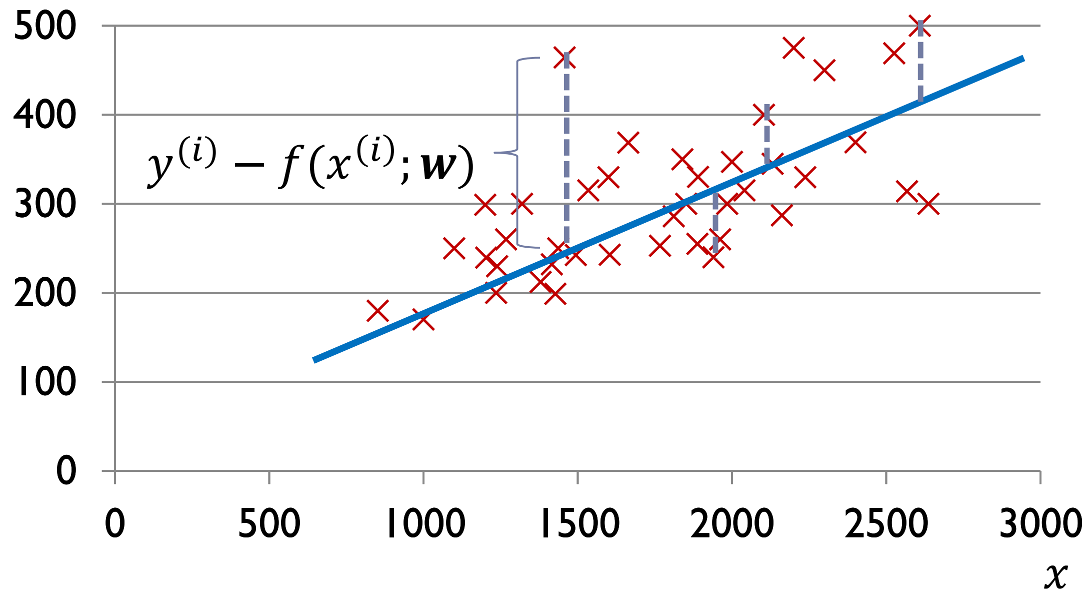
(2) choose $\boldsymbol{w}$ to minimize the error measure

How to measure the error



Squared error: $\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{w}\right)\right)^2$

Linear regression: univariate example



Cost function:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - f(x; \boldsymbol{w})\right)^2$$
$$= \sum_{i=1}^{n} \left(y^{(i)} - w_0 - w_1 x^{(i)}\right)^2$$

Regression: squared loss

- In the SSE cost function, we used squared error as the prediction loss:

$$Loss(y,\hat{y}) = (y-\hat{y})^2 \qquad \hat{y} = f(\boldsymbol{x};\boldsymbol{w})$$

- Cost function (based on the training set):

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} Loss\left(y^{(i)}, f\left(\boldsymbol{x}^{(i)};\boldsymbol{w}\right)\right)$$
$$= \sum_{i=1}^{n} \left(y^{(i)} - f\left(\boldsymbol{x}^{(i)};\boldsymbol{w}\right)\right)^2$$

- Minimizing sum (or mean) of squared errors is a common approach in curve fitting, neural network, etc.

Sum of Squares Error (SSE) cost function

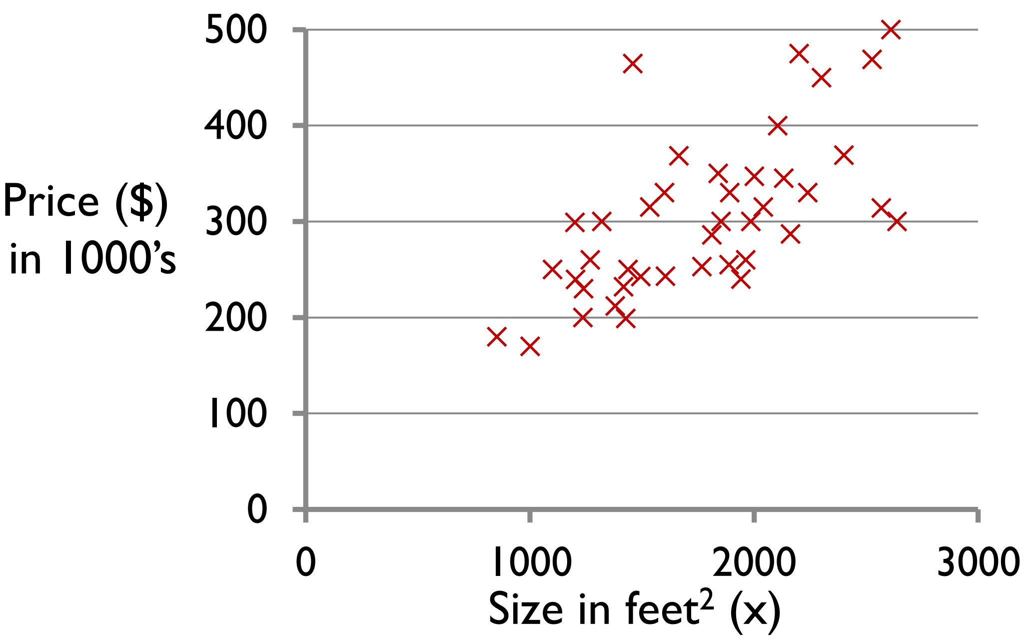
$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\right)^2$$

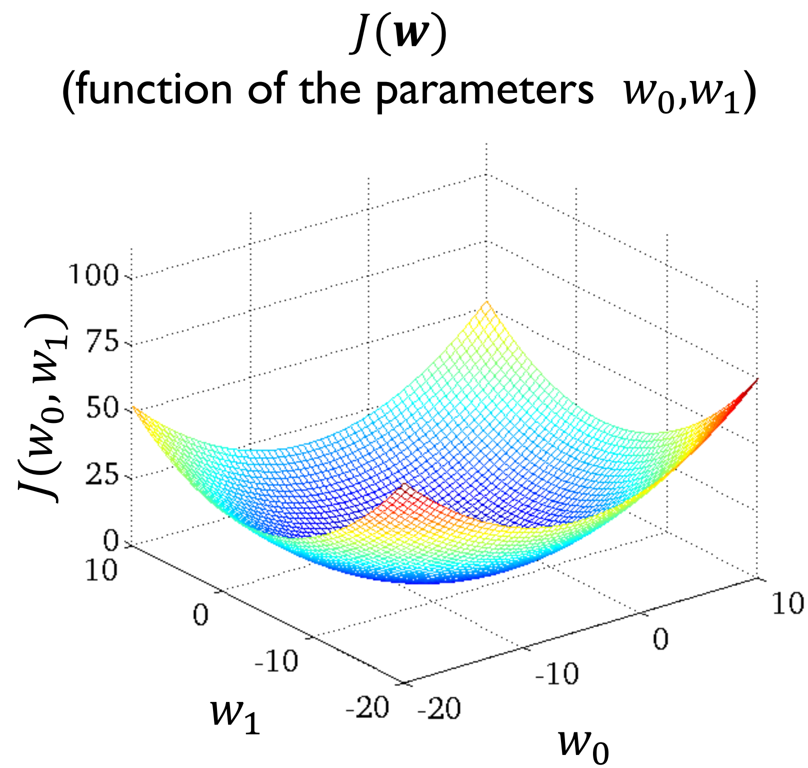
- $J(\boldsymbol{w})$: sum of the squares of the prediction errors on the training set

- We want to find the best regression function $f\left(\boldsymbol{x}^{(i)}; \boldsymbol{w}\right)$
  - equivalently, the best $\boldsymbol{w}$

- Minimize $J(\boldsymbol{w})$
  - Find optimal $\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}; \widehat{\boldsymbol{w}})$ where $\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$

Cost function: univariate example



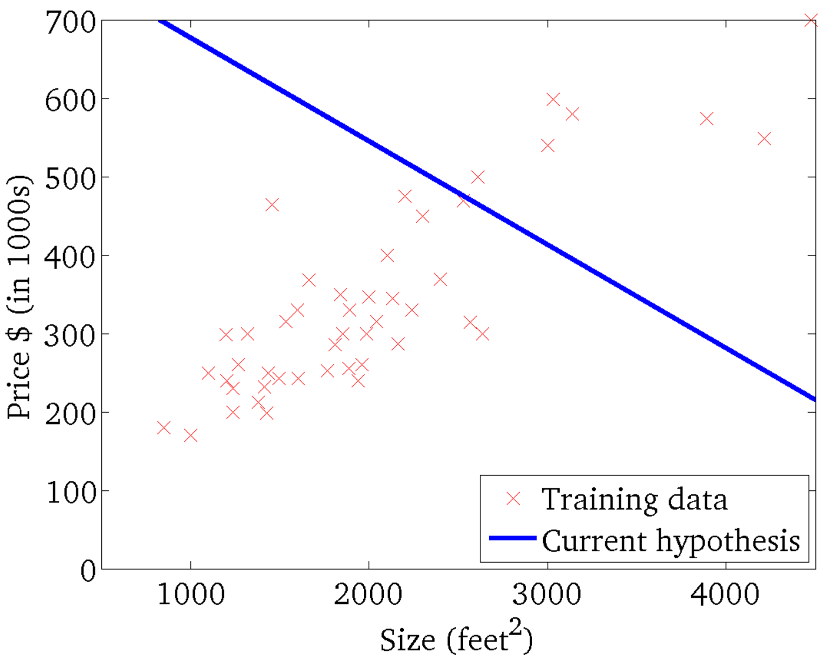


This example has been adapted from: Prof. Andrew Ng's slides

Cost function: univariate example

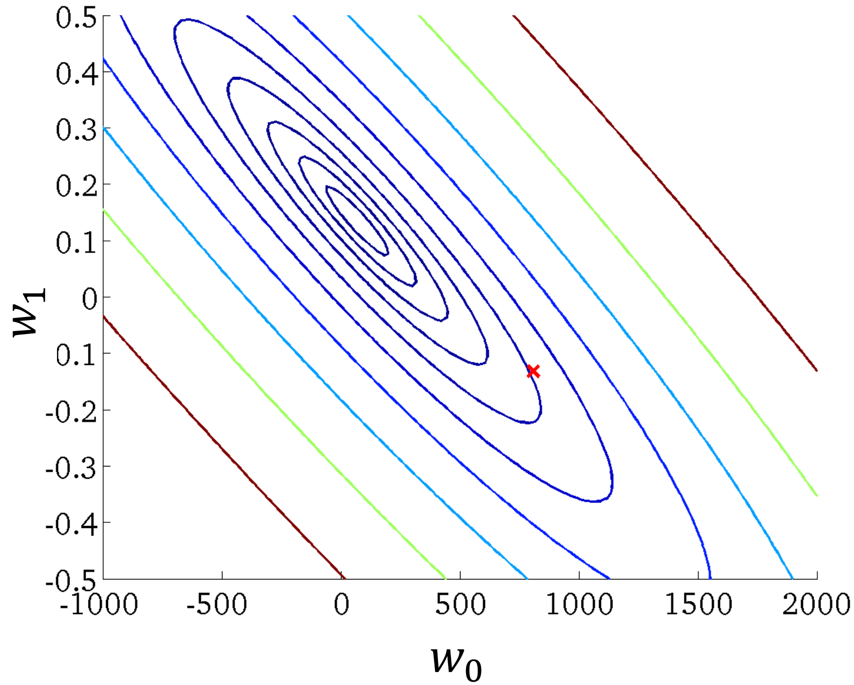
$$f(x; w_0, w_1) = w_0 + w_1 x$$

(for fixed w_0,w_1, this is a function of x)



$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)



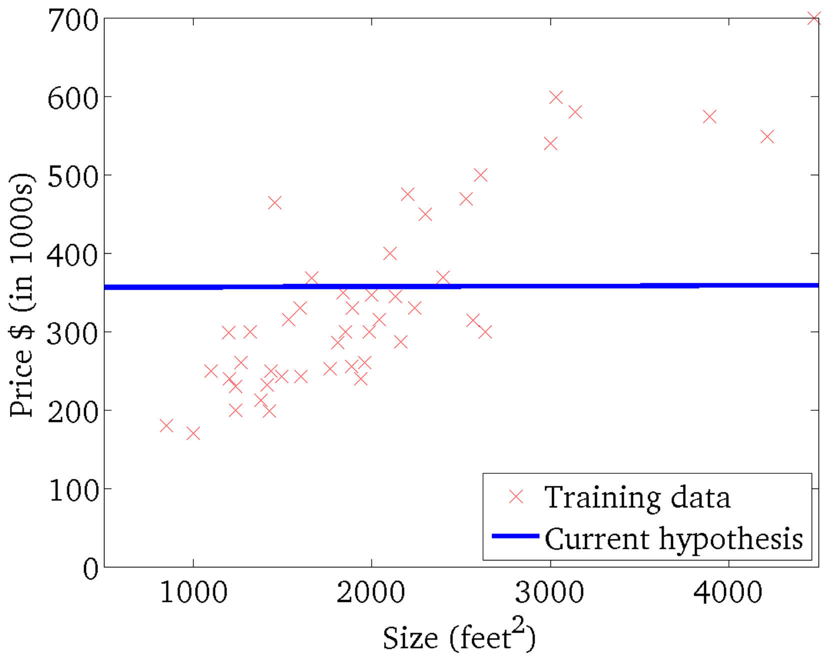
This example has been adapted from: Prof. Andrew Ng's slides

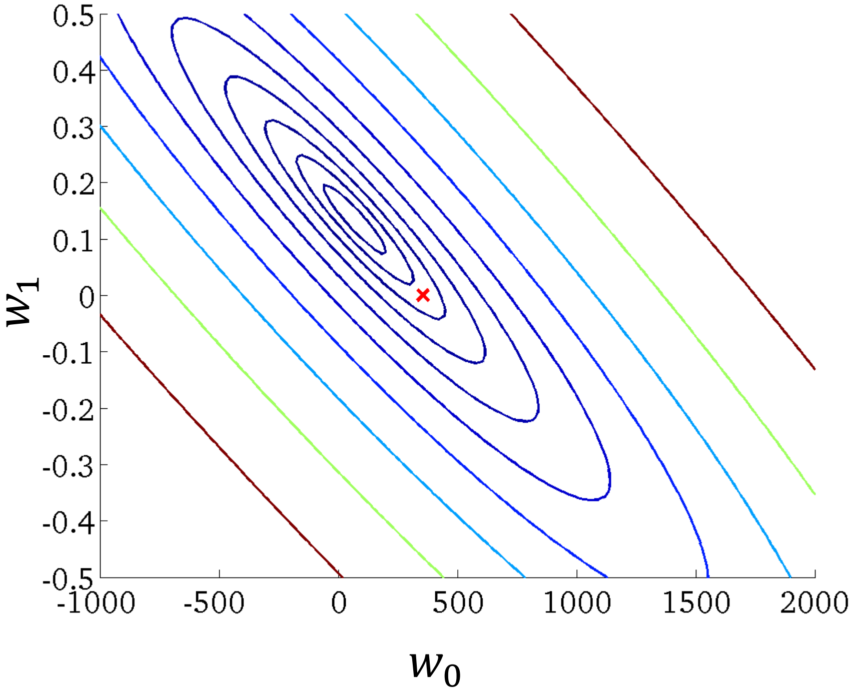
Cost function: univariate example

$$f(x; w_0, w_1) = w_0 + w_1 x$$

$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)

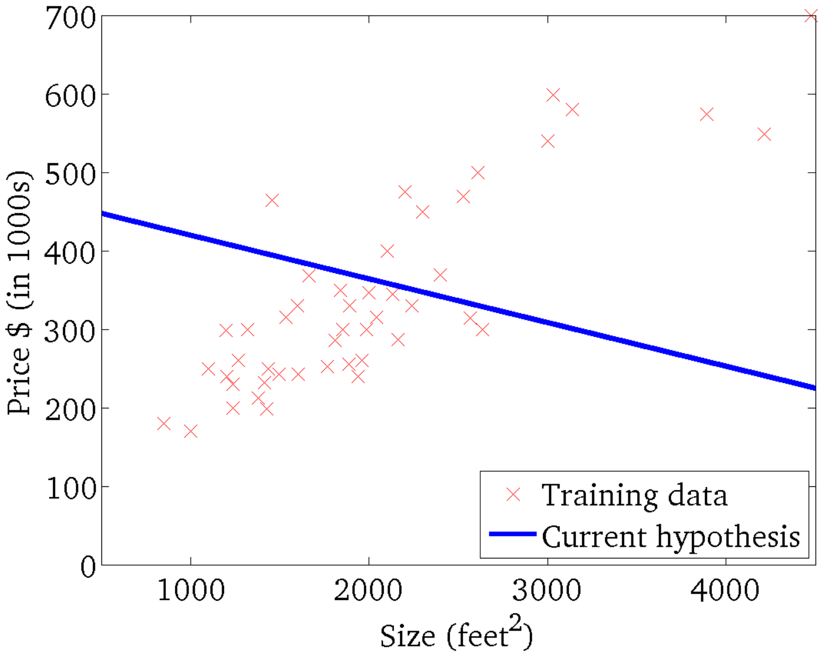




This example has been adapted from: Prof. Andrew Ng's slides

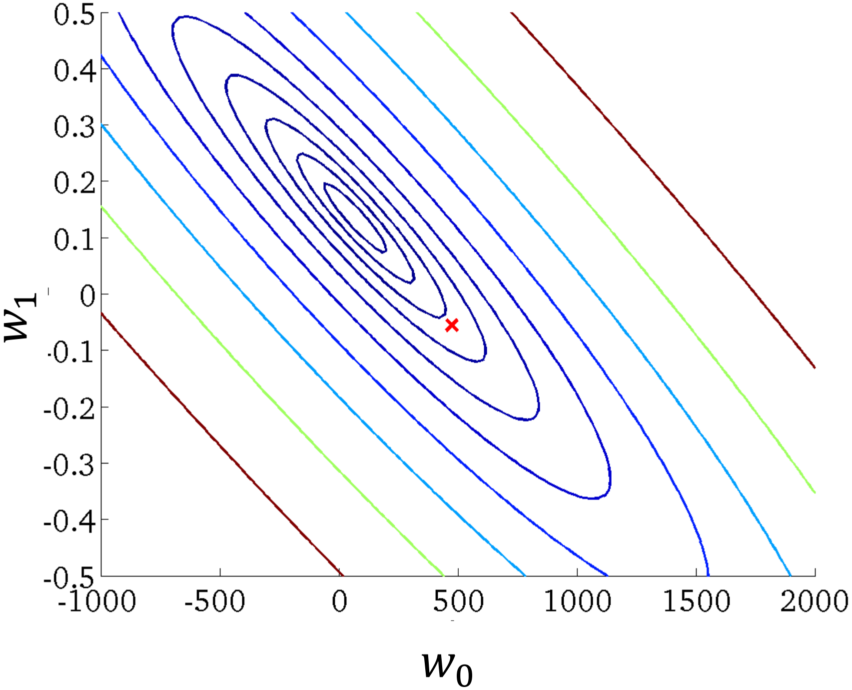
Cost function: univariate example

$f(x; w_0, w_1) = w_0 + w_1 x$



$J(w_0, w_1)$

(function of the parameters w_0,w_1)



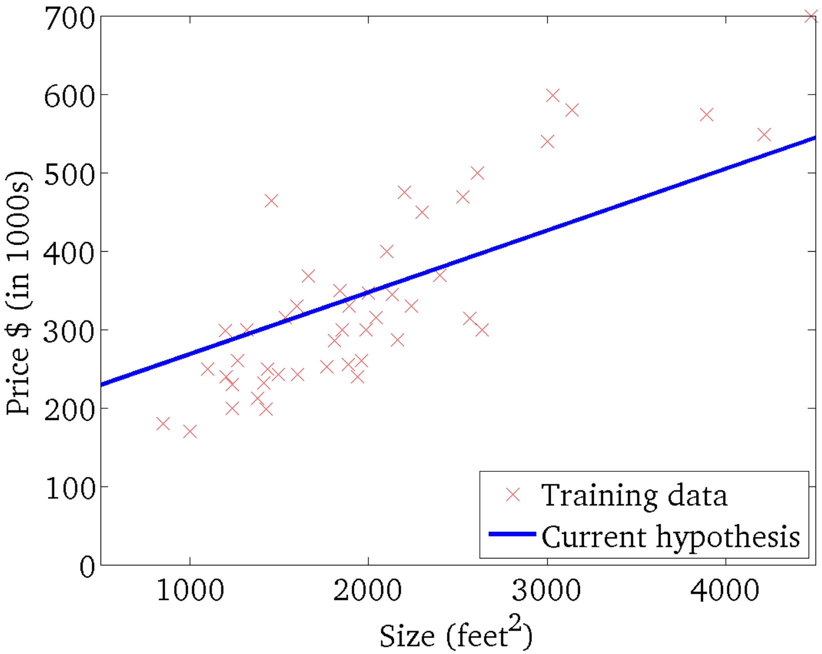
This example has been adapted from: Prof. Andrew Ng's slides

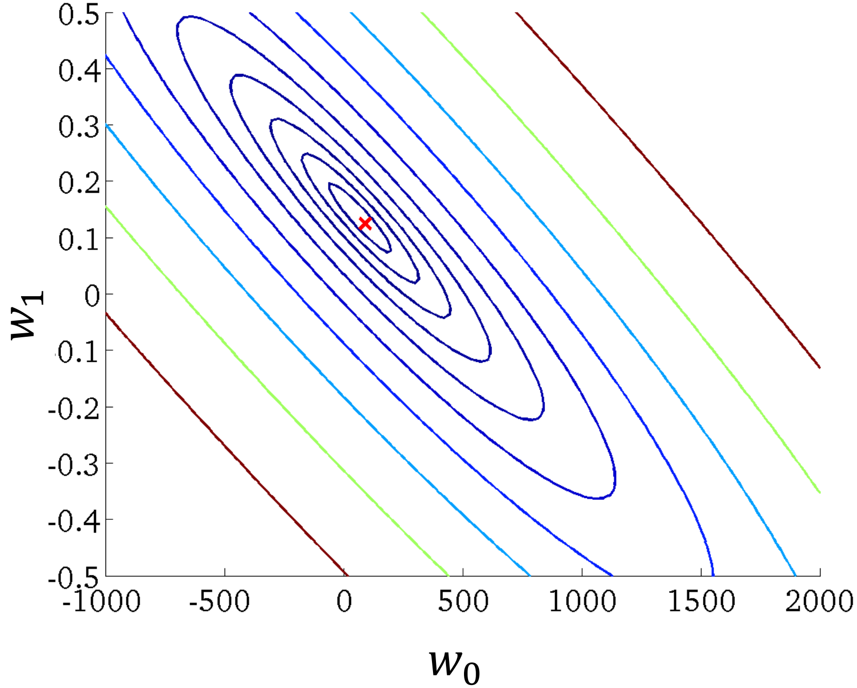
Cost function: univariate example

$f(x; w_0, w_1) = w_0 + w_1 x$

$J(w_0, w_1)$

(function of the parameters w_0,w_1)





 This example has been adapted from: Prof. Andrew Ng's slides

Cost function optimization: univariate

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - w_0 - w_1 x^{(i)}\right)^2$$

- Necessary conditions for the "optimal" parameter values:

$$\frac{\partial J(\boldsymbol{w})}{\partial w_0} = 0$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_1} = 0$$

Optimality conditions: univariate

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - w_0 - w_1 x^{(i)}\right)^2$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_1} = \sum_{i=1}^{n} 2\left(y^{(i)} - w_0 - w_1 x^{(i)}\right)\left(-x^{(i)}\right) = 0$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_0} = \sum_{i=1}^{n} 2\left(y^{(i)} - w_0 - w_1 x^{(i)}\right)(-1) = 0$$

- A systems of 2 linear equations

Cost function: multivariate

- We have to minimize the empirical squared loss:

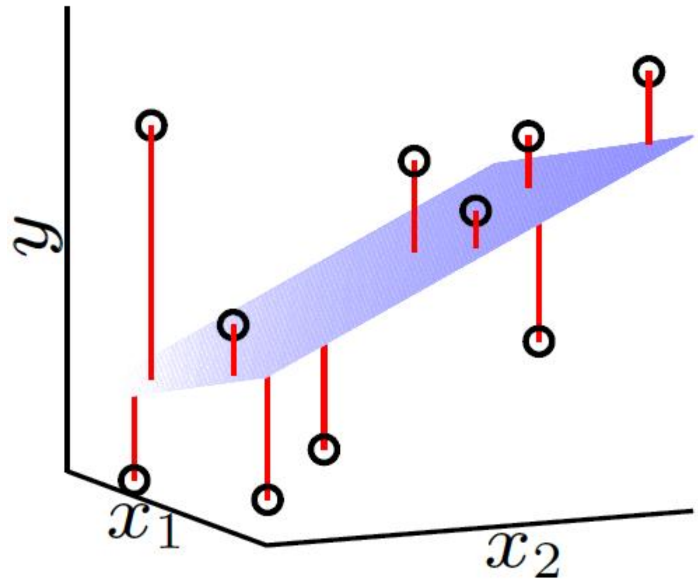
$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\right)^2$$

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x_1 + \dots w_d x_d$$

$$\boldsymbol{w} = [w_0, w_1, \dots, w_d]^T$$

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} J(\boldsymbol{w})$$

Cost function and optimal linear model



- Necessary conditions for the "optimal" parameter values:

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \mathbf{0}$$

- A system of $d + 1$ linear equations

Cost function: matrix notation

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\right)^2 =$$

$$= \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2$$

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$

$$J(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2$$

Minimizing cost function

Optimal linear weight vector (for SSE cost function):

$$J(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2$$

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$$

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \boldsymbol{0} \Rightarrow \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}$$

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

Minimizing cost function

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

$$\boldsymbol{w} = \boldsymbol{X}^{\dagger}\boldsymbol{y}$$

$$\boldsymbol{X}^{\dagger} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$$

$\boldsymbol{X}^{\dagger}$ is pseudo inverse of $\boldsymbol{X}$

Another approach for optimizing the sum squared error

- Iterative approach for solving the following optimization problem:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w}) \right)^2$$

Review: Iterative optimization of cost function

- Cost function: $J(\boldsymbol{w})$
- Optimization problem: $\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argm}in}\, J(\boldsymbol{w})$
- Steps:
  - Start from $\boldsymbol{w}^0$
  - Repeat
    - Update $\boldsymbol{w}^t$ to $\boldsymbol{w}^{t+1}$ in order to reduce J
    - $t \leftarrow t + 1$
  - until we hopefully end up at a minimum

Review: Gradient descent

- First-order optimization algorithm to find $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$
  - Also known as "**steepest descent**"

- In each step, takes steps proportional to the negative of the gradient vector of the function at the current point $\boldsymbol{w}^t$:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \gamma_t \nabla J(\boldsymbol{w}^t)$$

  - $J(\boldsymbol{w})$ decreases fastest if one goes from $\boldsymbol{w}^t$ in the direction of $-\nabla J(\boldsymbol{w}^t)$
  - Assumption: $J(\boldsymbol{w})$ is defined and differentiable in a neighborhood of a point $\boldsymbol{w}^t$

Gradient ascent takes steps proportional to (the positive of) the gradient to find a local maximum of the function

Review: Gradient descent

- Minimize $J(\boldsymbol{w})$

Step size
(Learning rate parameter)

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{t})$$

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = [\frac{\partial J(\boldsymbol{w})}{\partial w_1}, \frac{\partial J(\boldsymbol{w})}{\partial w_2}, \dots, \frac{\partial J(\boldsymbol{w})}{\partial w_d}]$$

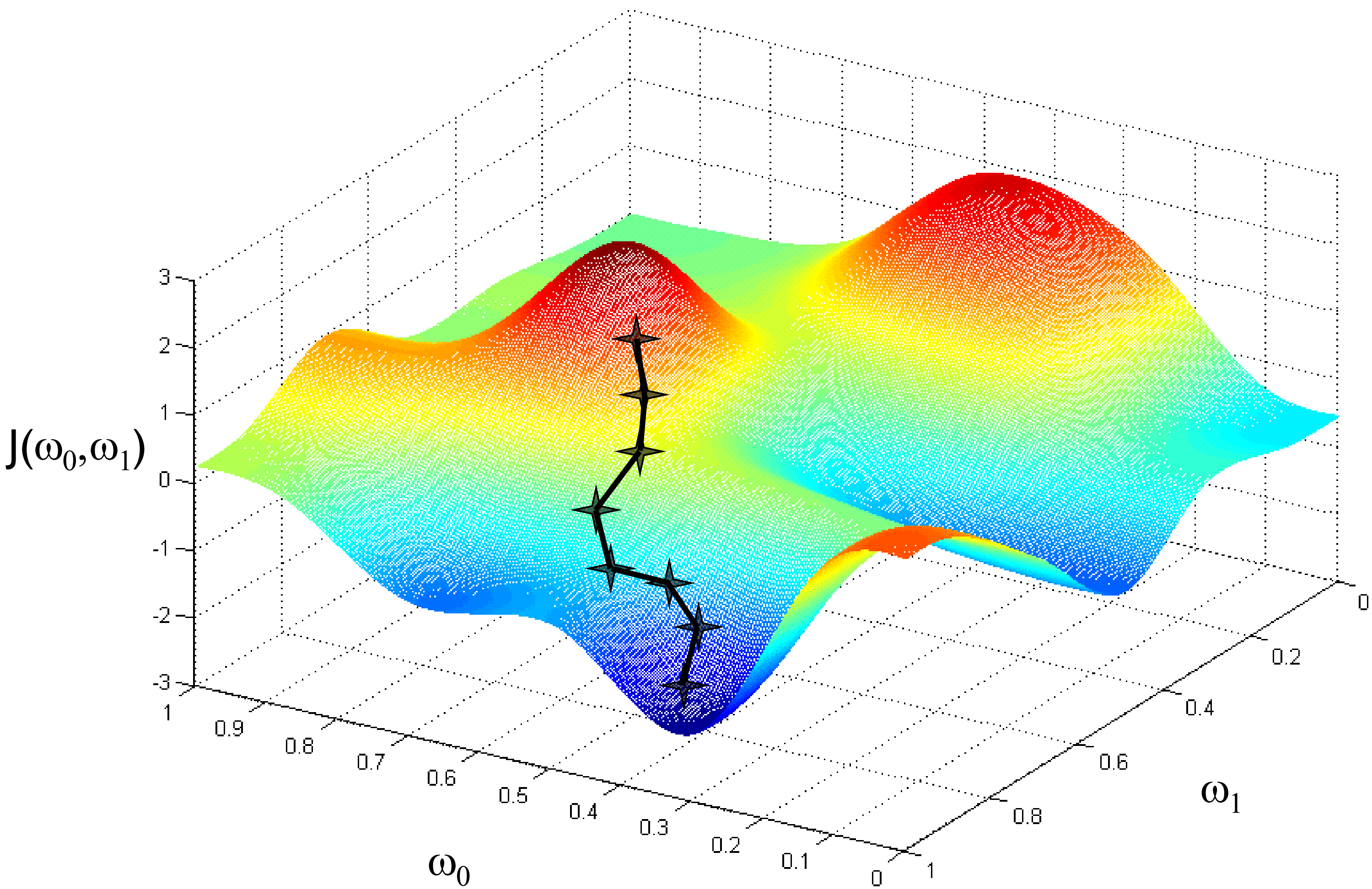
- If η is small enough, then $J(\boldsymbol{w}^{t+1}) \leq J(\boldsymbol{w}^{t})$.
- η can be allowed to change at every iteration as η_t.

Review: Gradient descent disadvantages

- Local minima problem

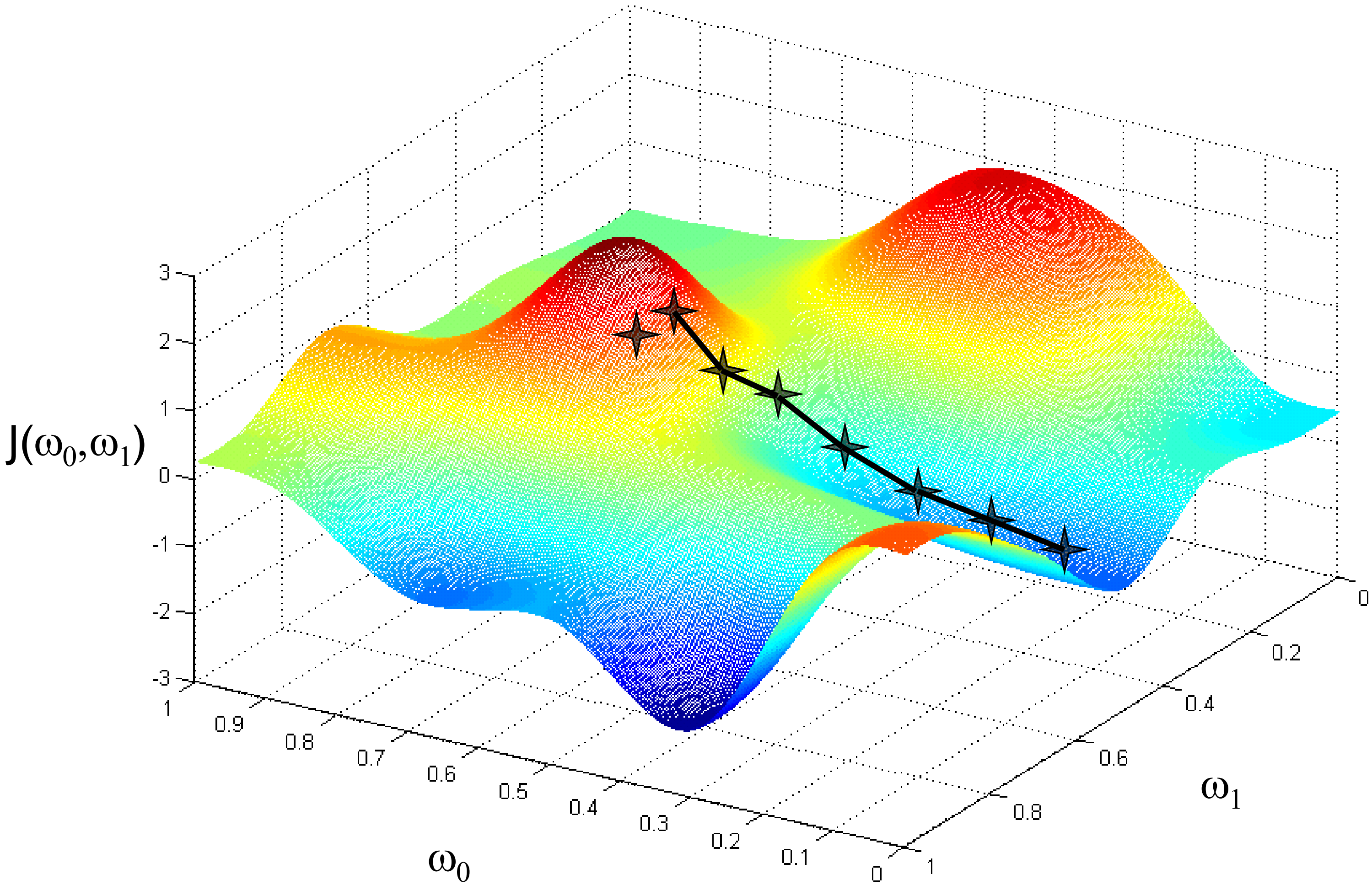
- However, when J is convex, all local minima are also global minima $\Rightarrow$ gradient descent can converge to the global solution.

Review: Problem of gradient descent with non-convex cost functions



This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

Review: Problem of gradient descent with non-convex cost functions



This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

Gradient descent for SSE cost function

- Minimize $J(\boldsymbol{w})$

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{t})$$

- $J(\boldsymbol{w})$: Sum of squares error

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - f\left(\boldsymbol{x}^{(i)}; \boldsymbol{w}\right) \right)^2$$

- Weight update rule for $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \eta \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{w}^{t^T} \boldsymbol{x}^{(i)} \right) \boldsymbol{x}^{(i)}$$

Gradient descent for SSE cost function

- Weight update rule: $f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x}$

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)\boldsymbol{x}^{(i)}$$

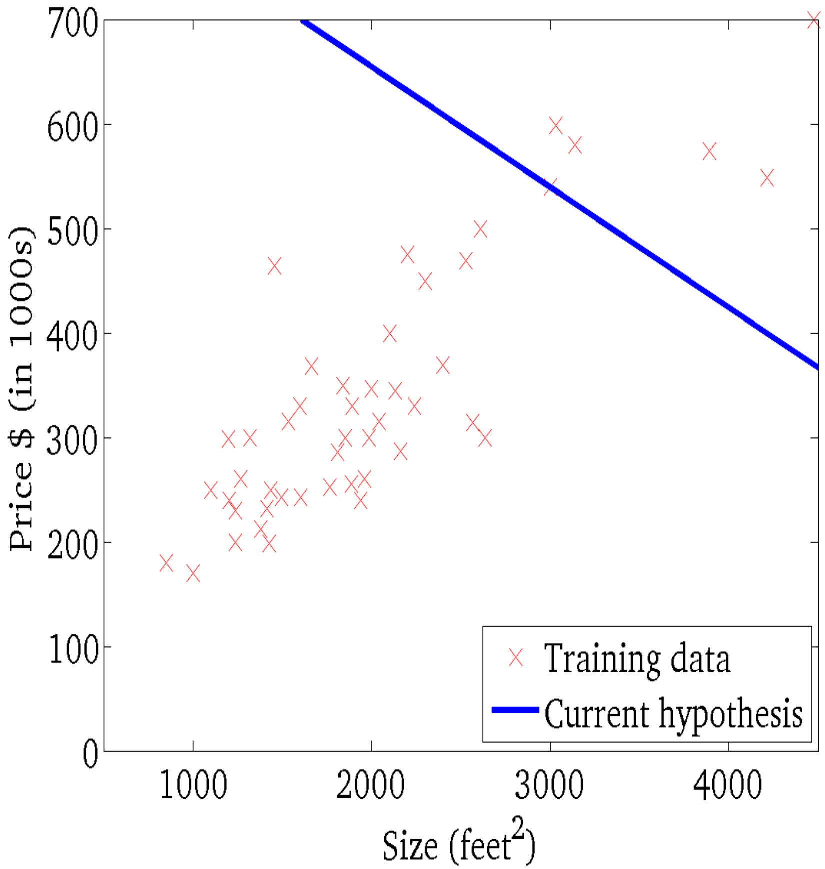
Batch mode: each step considers all training data

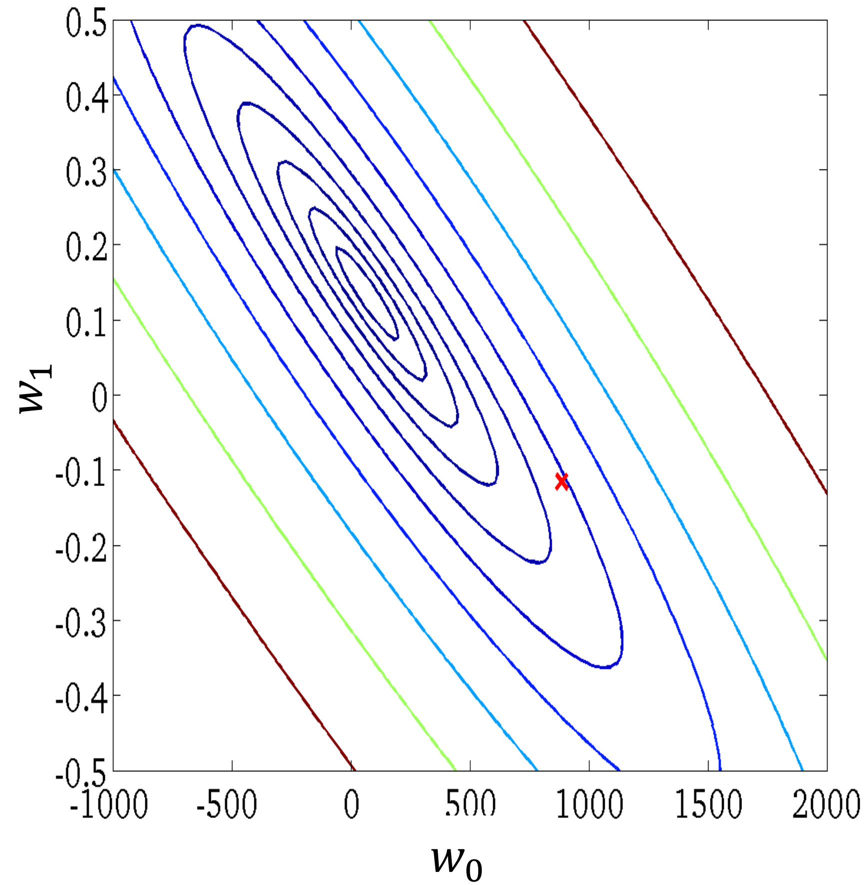
- η: too small → gradient descent can be slow.
- η: too large → gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

$$f(x; w_0, w_1) = w_0 + w_1 x$$

$J(w_0, w_1)$

(function of the parameters w_0,w_1)



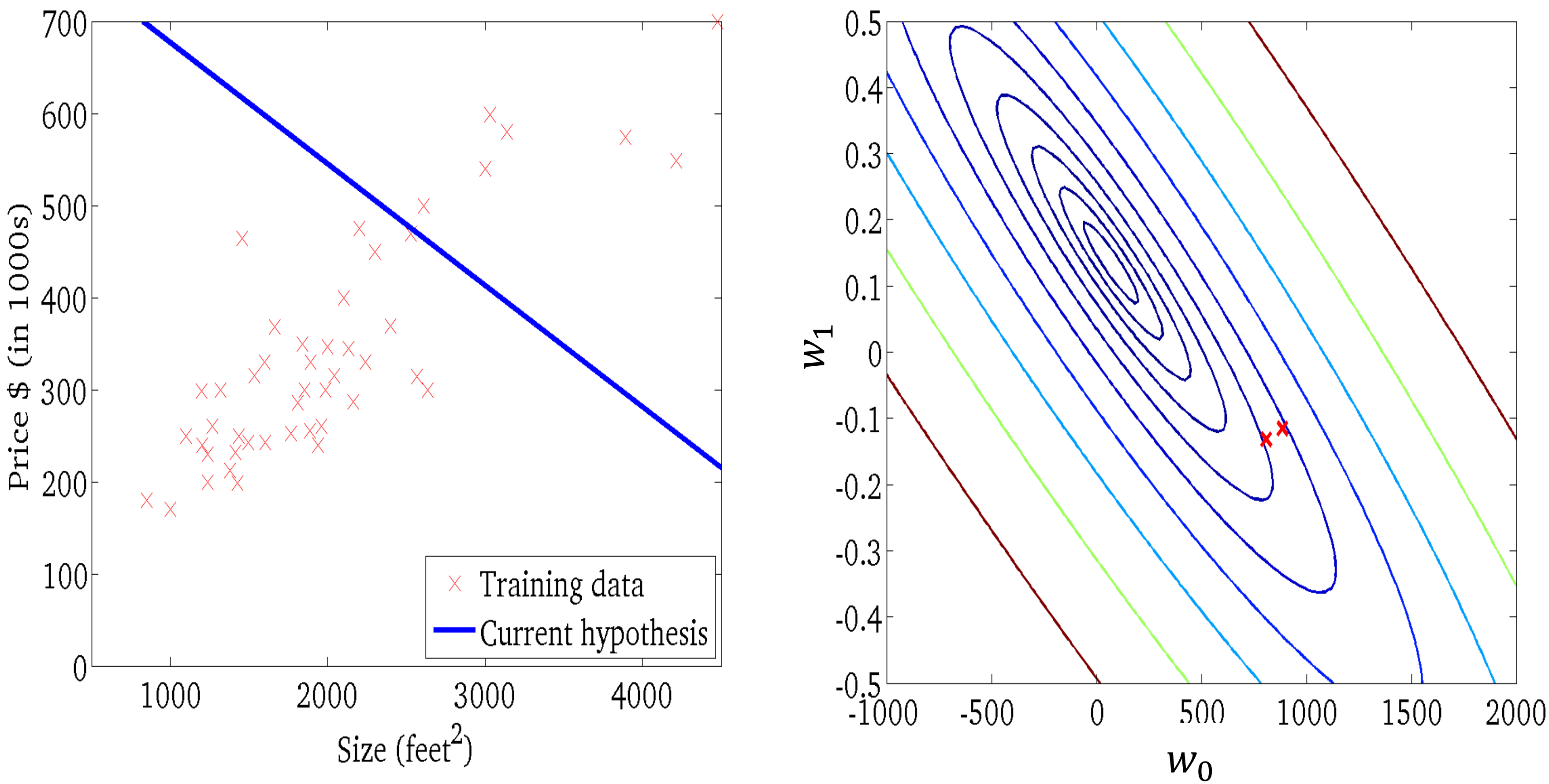


This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

$$f(x; w_0, w_1) = w_0 + w_1 x$$

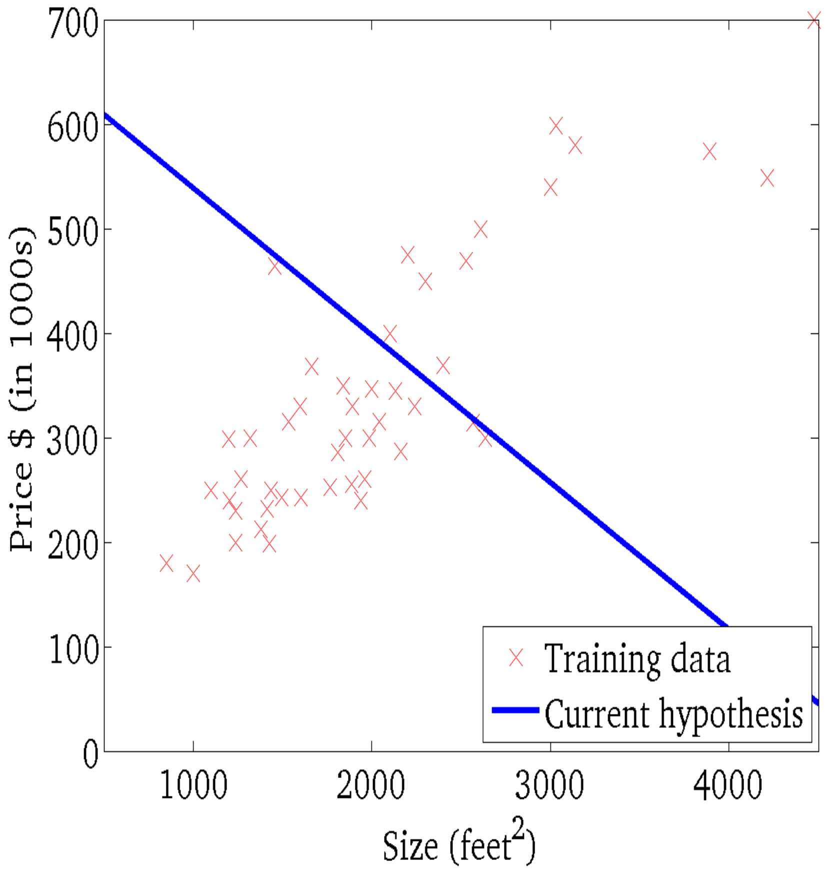
$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)



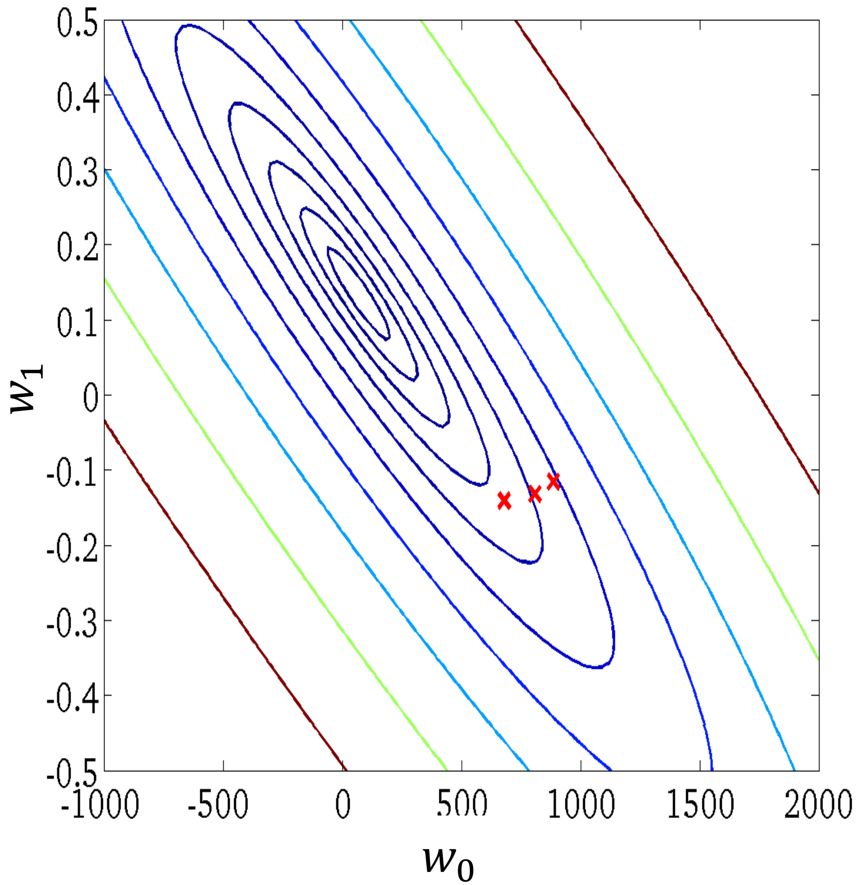
 This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

$$f(x; w_0, w_1) = w_0 + w_1 x$$



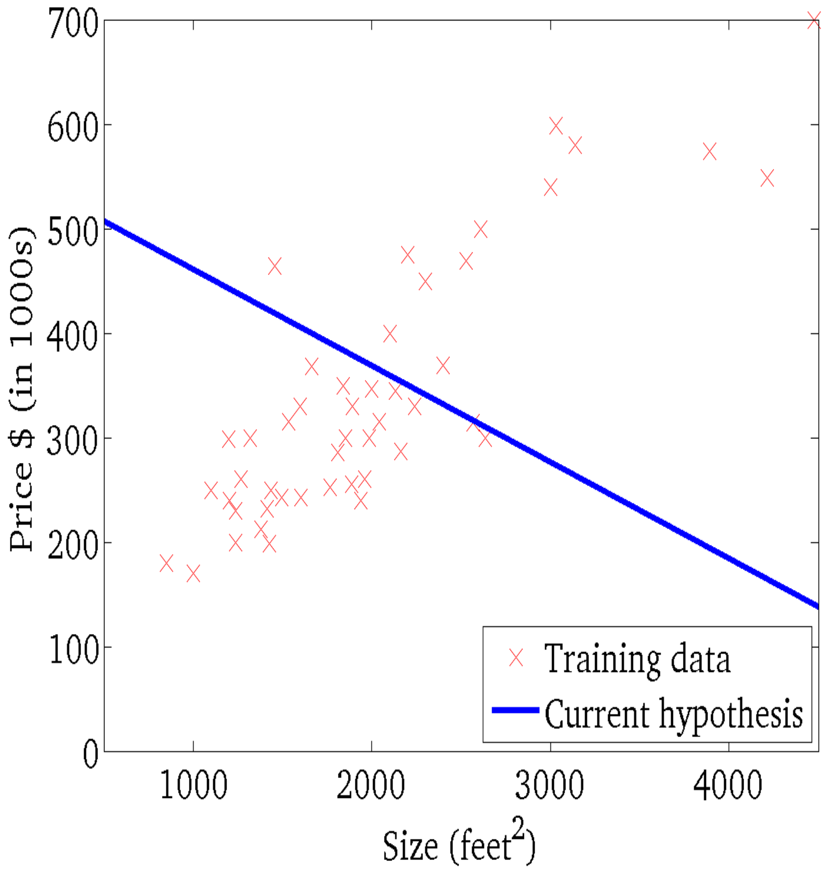
$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)



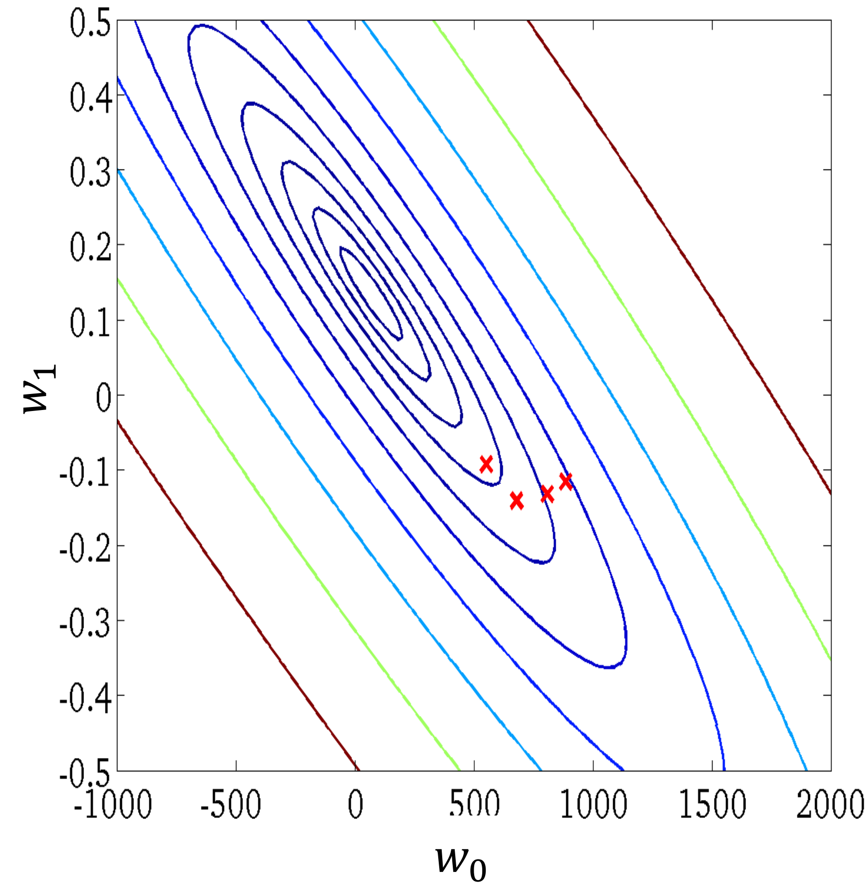
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

$$f(x; w_0, w_1) = w_0 + w_1 x$$

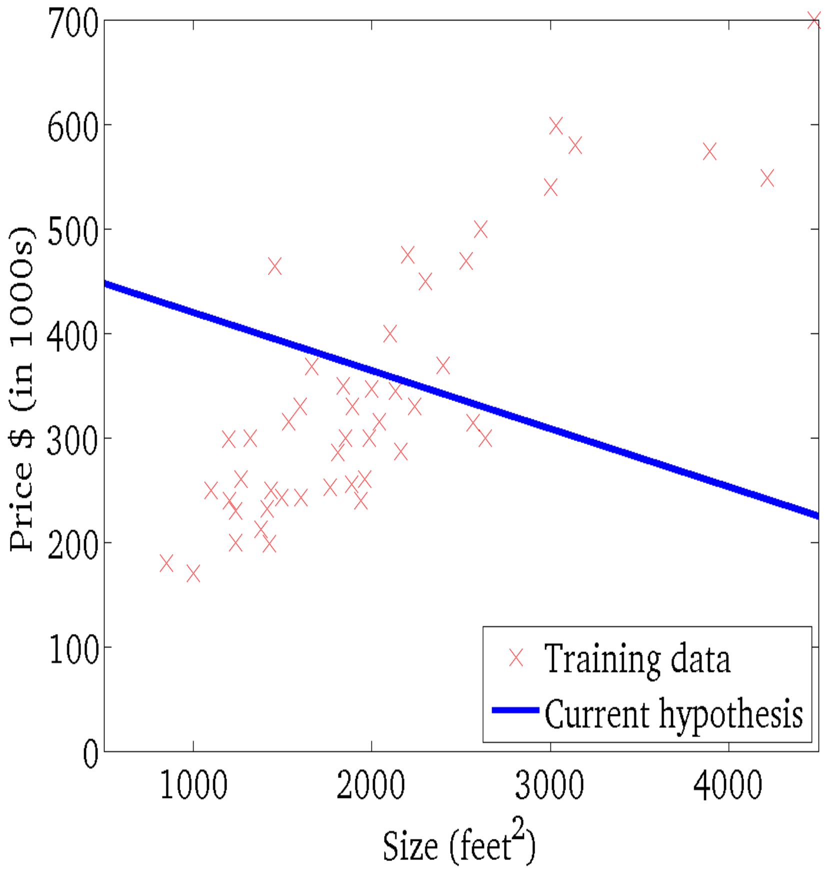


$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)

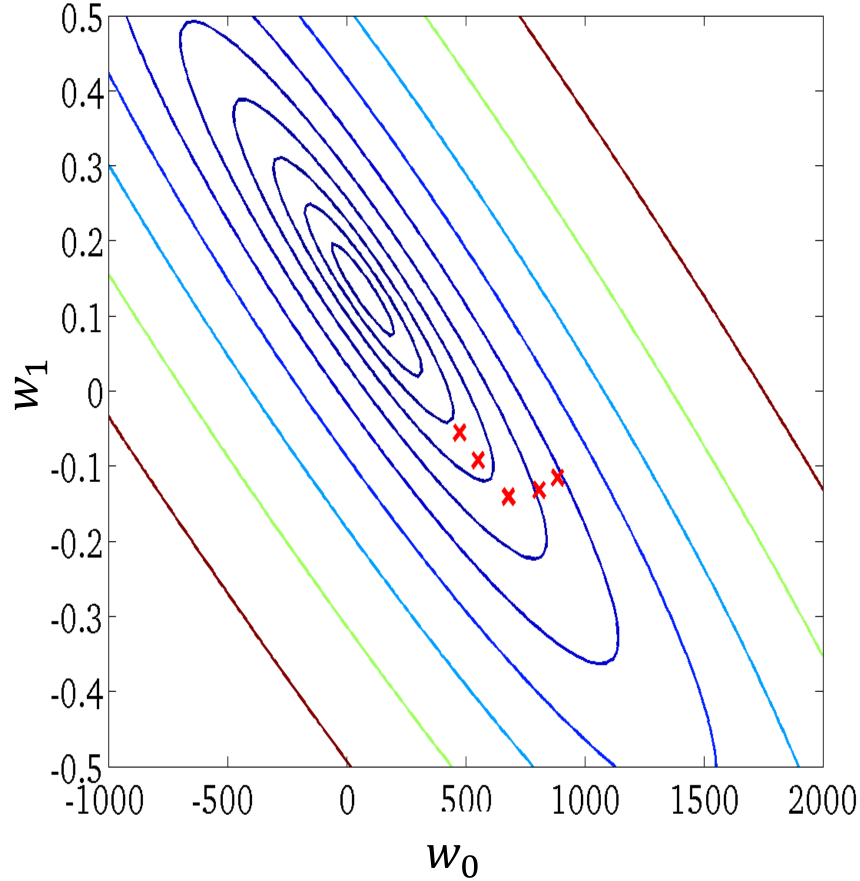


$$f(x; w_0, w_1) = w_0 + w_1 x$$



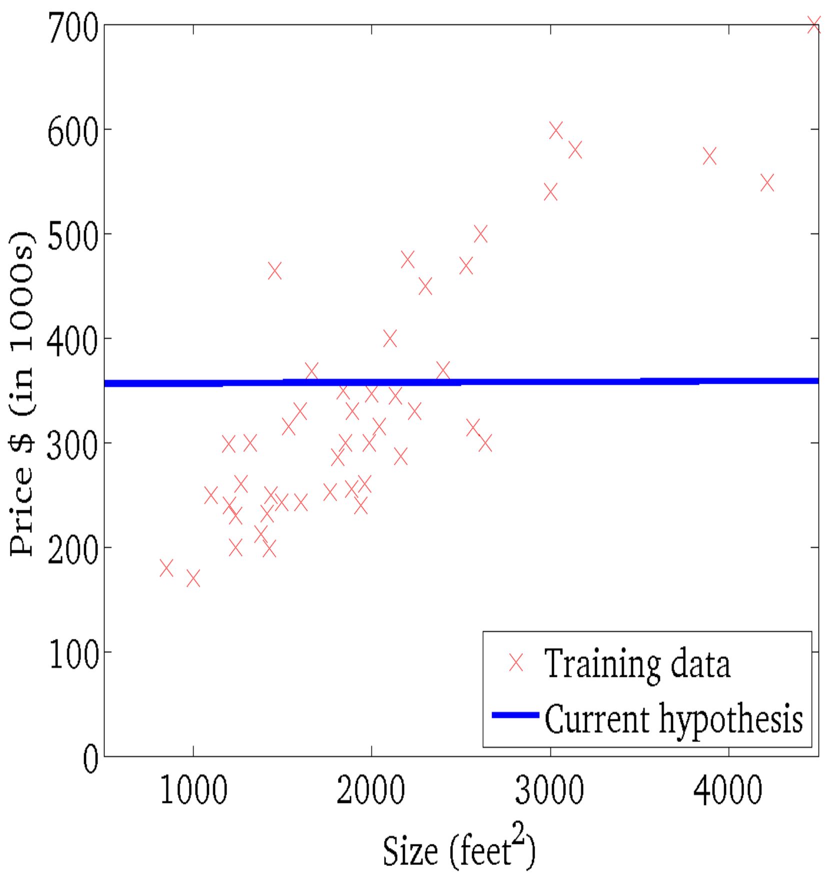
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



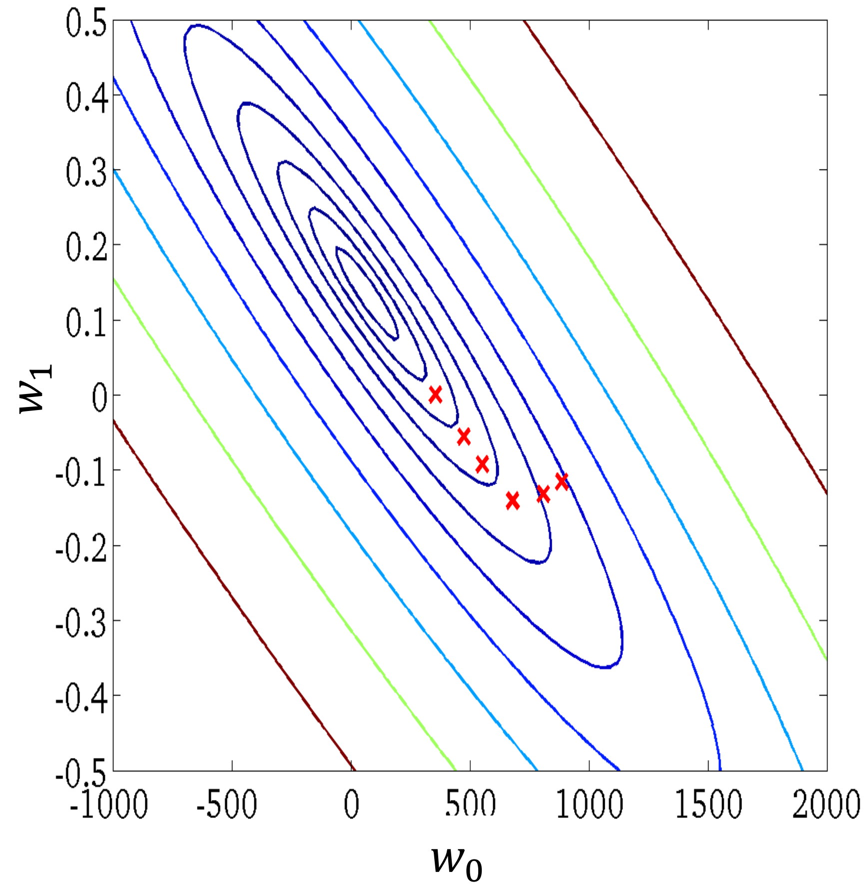
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

$$f(x; w_0, w_1) = w_0 + w_1 x$$



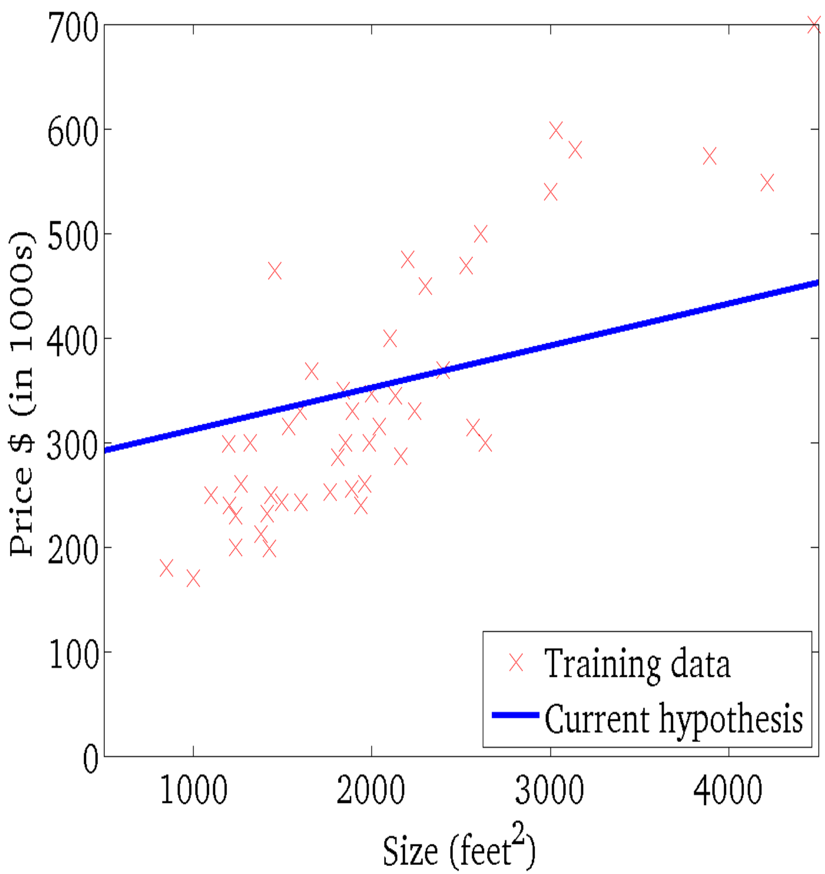
$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)



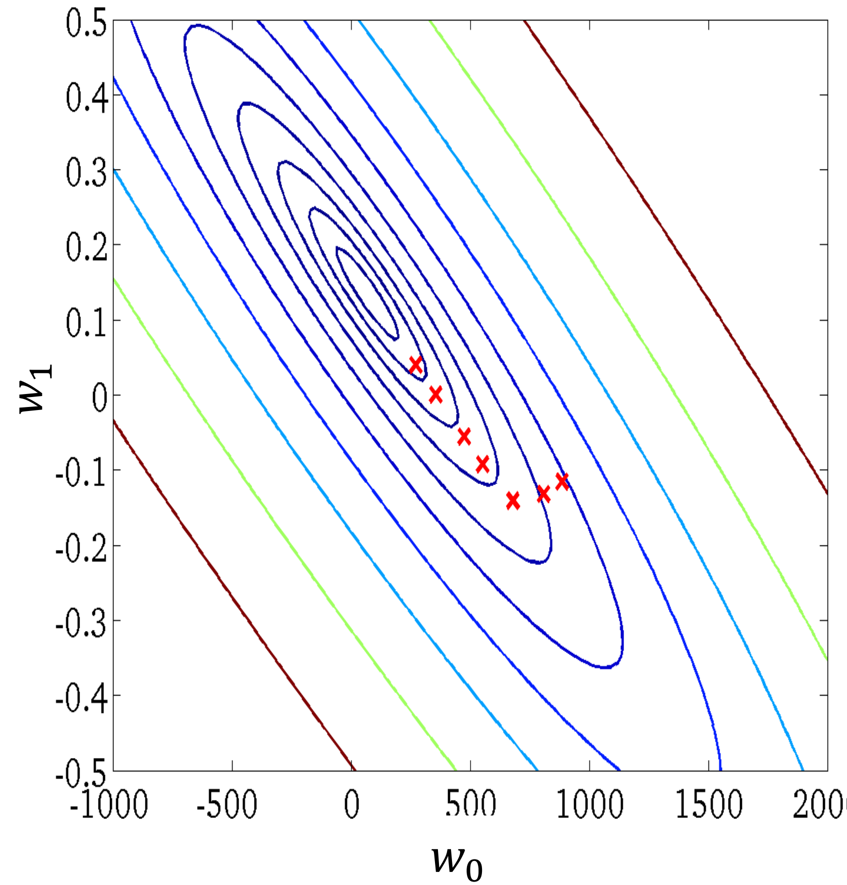
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

$f(x; w_0, w_1) = w_0 + w_1 x$



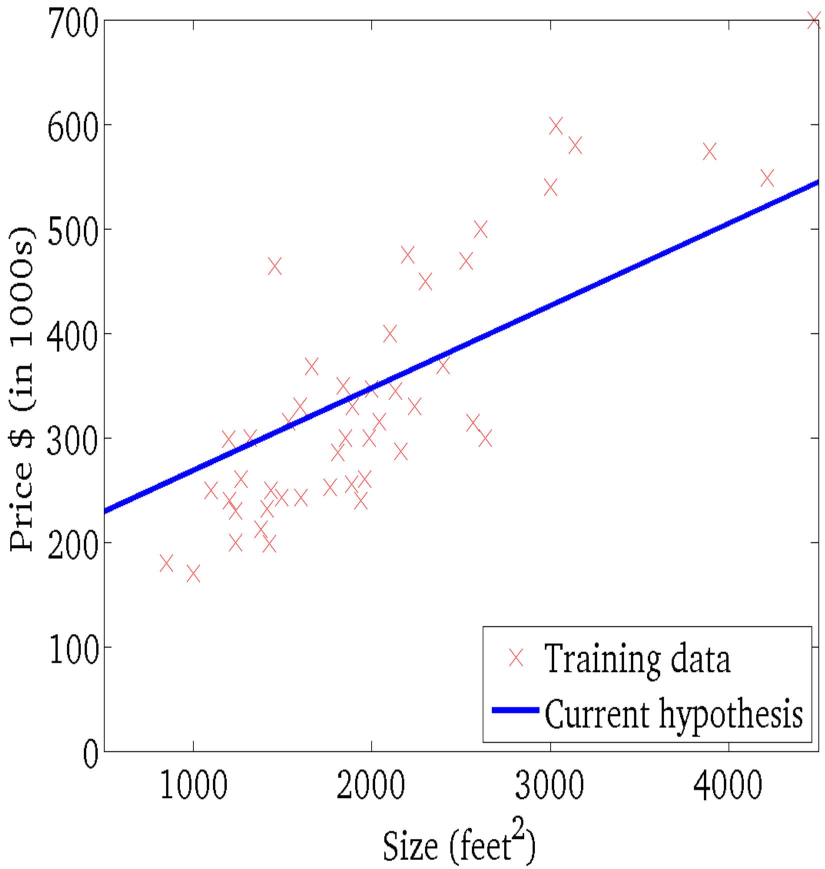
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



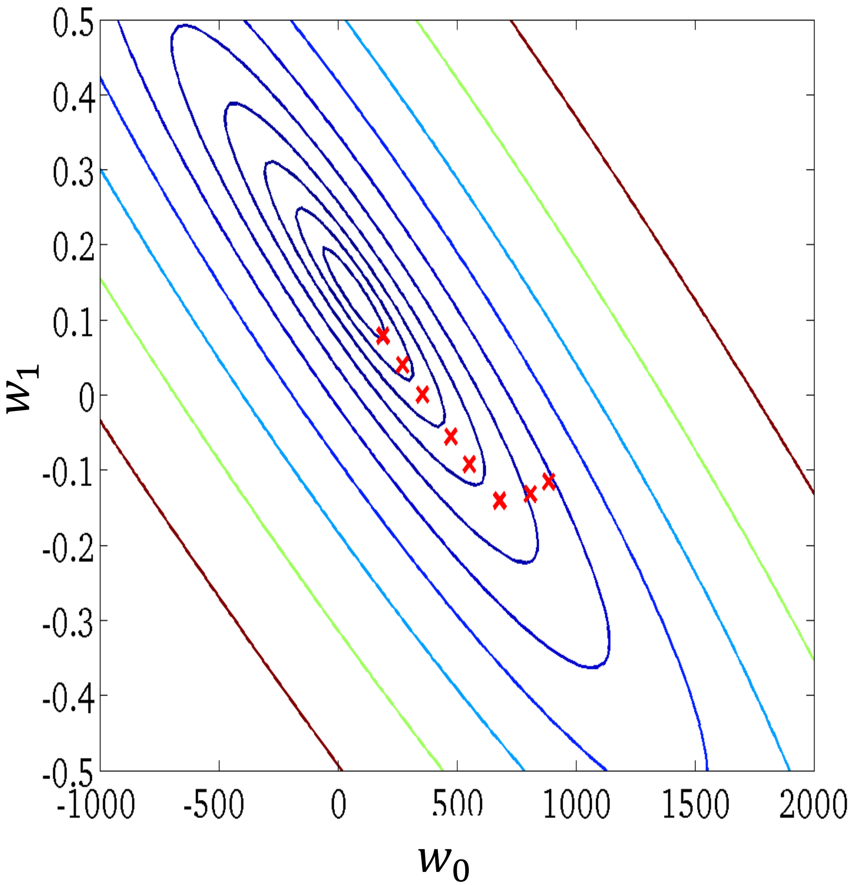
 This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

$f(x; w_0, w_1) = w_0 + w_1 x$



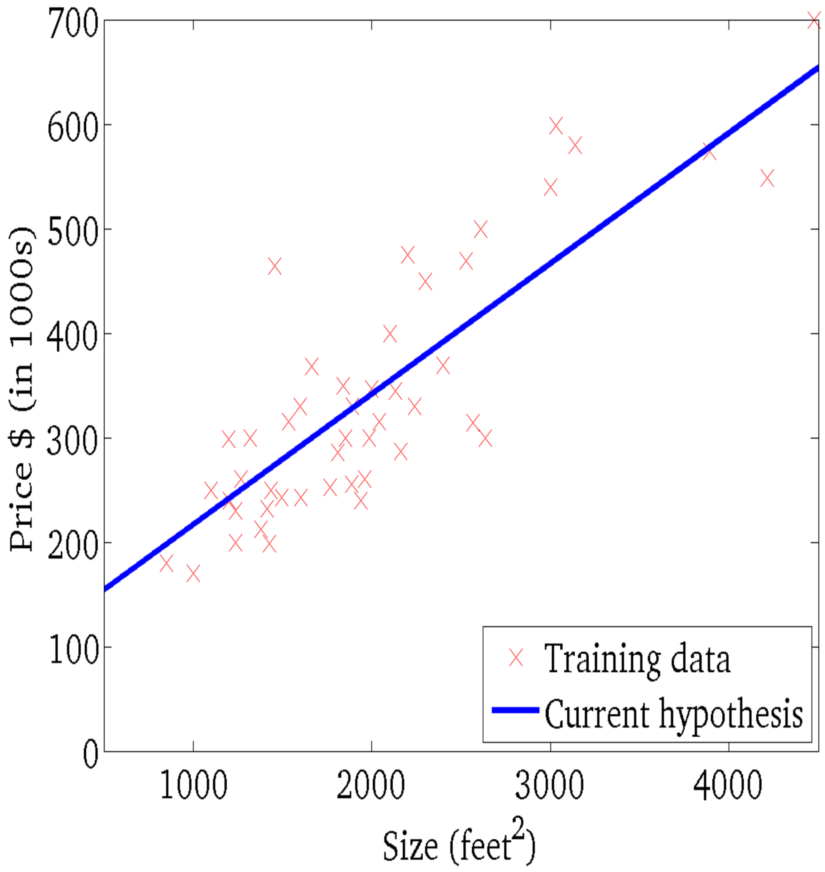
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



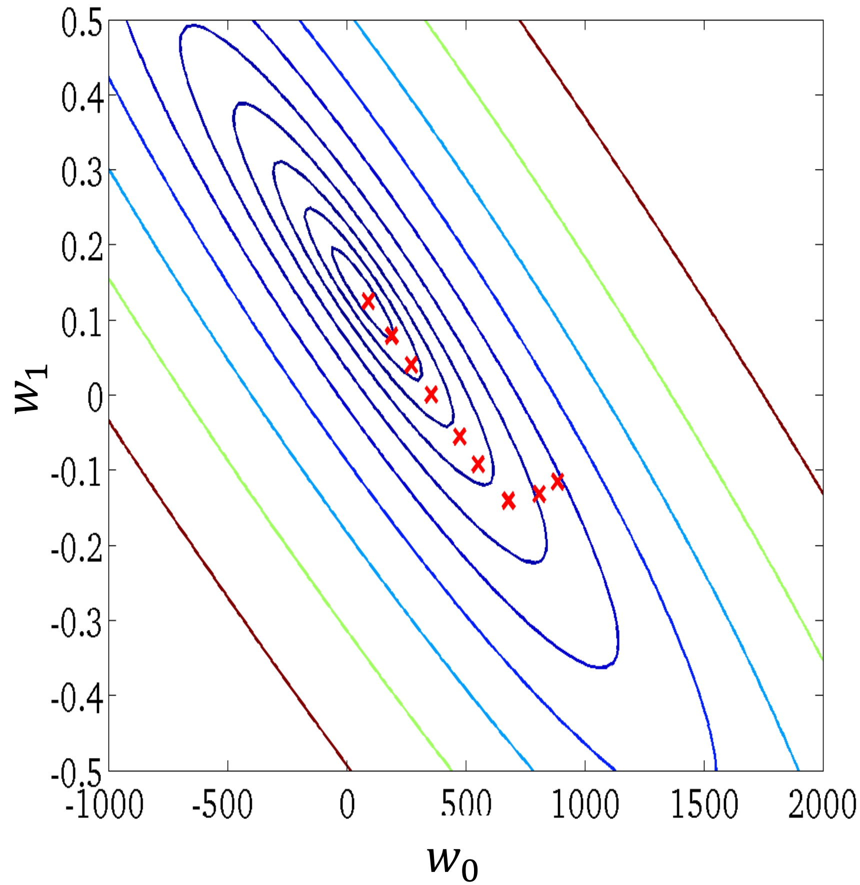
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

$$f(x; w_0, w_1) = w_0 + w_1 x$$



$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)



This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

Stochastic gradient descent

- Batch techniques process the entire training set in one go
  - thus they can be computationally costly for large data sets.

- Stochastic gradient descent: when the cost function can comprise a sum over data points:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} J^{(i)}(\boldsymbol{w})$$

- Update after presentation of $(\boldsymbol{x}^{(i)}, y^{(i)})$:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \eta \nabla_{\boldsymbol{w}} J^{(i)}(\boldsymbol{w})$$

Stochastic gradient descent

- Example: Linear regression with SSE cost function

$$J^{(i)}(\boldsymbol{w}) = \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2$$

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \nabla_{\boldsymbol{w}} J^{(i)}(\boldsymbol{w})$$

$$\boxed{\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta\left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)\boldsymbol{x}^{(i)}}$$

Least Mean Squares (LMS)

It is proper for sequential or online learning

Stochastic gradient descent: online learning

- Sequential learning is also appropriate for real-time applications
  - data observations are arriving in a continuous stream
  - and predictions must be made before seeing all of the data

- The value of η needs to be chosen with care to ensure that the algorithm converges

Evaluation and generalization

- Why minimizing the cost function (based on only training data) while we are interested in the performance on new examples?

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} Loss\left(y^{(i)}, f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})\right) \longrightarrow \text{Empirical loss}$$

- **Evaluation**: After training, we need to measure how well the learned prediction function can predicts the target for unseen examples

Training and test performance

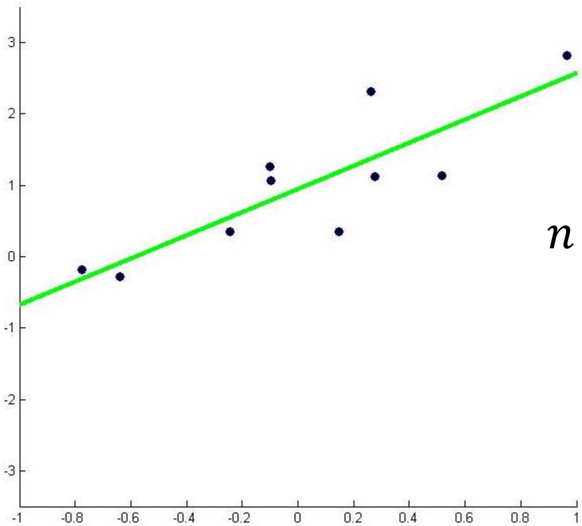
- Assumption: training and test examples are drawn independently at random from the same but unknown distribution.
  - Each training/test example $(\boldsymbol{x}, y)$ is a sample from joint probability distribution $P(\boldsymbol{x}, y)$, i.e., $(\boldsymbol{x}, y) \sim P$

Empirical (training) loss = $\frac{1}{n}\sum_{i=1}^{n} Loss\left(y^{(i)}, f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})\right)$

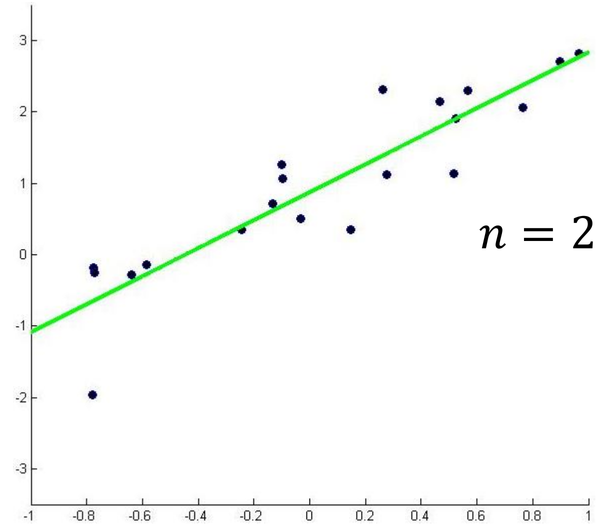
Expected (test) loss = $E_{\boldsymbol{x},y}\{Loss(y, f(\boldsymbol{x}; \boldsymbol{\theta}))\}$

- We minimize empirical loss (on the training data) and expect to also find an acceptable expected loss
  - Empirical loss as a proxy for the performance over the whole distribution.

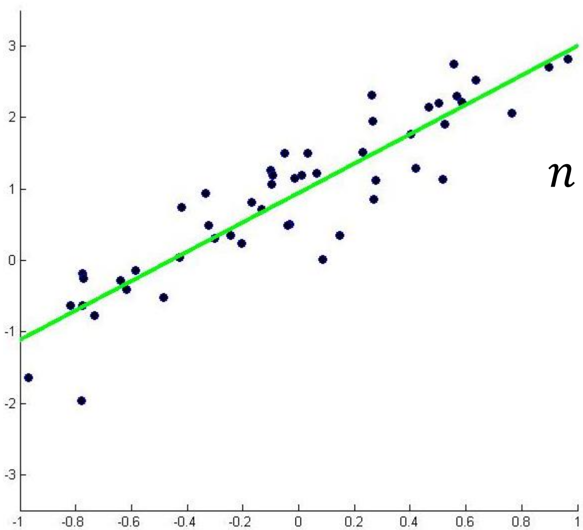
Linear regression: number of training data



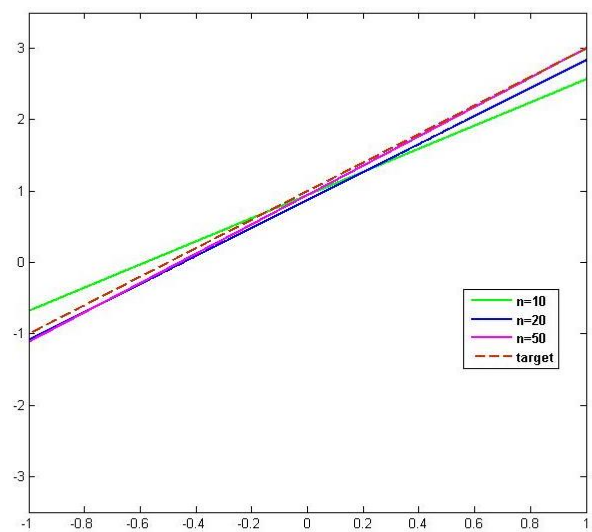
$n = 10$



$n = 20$

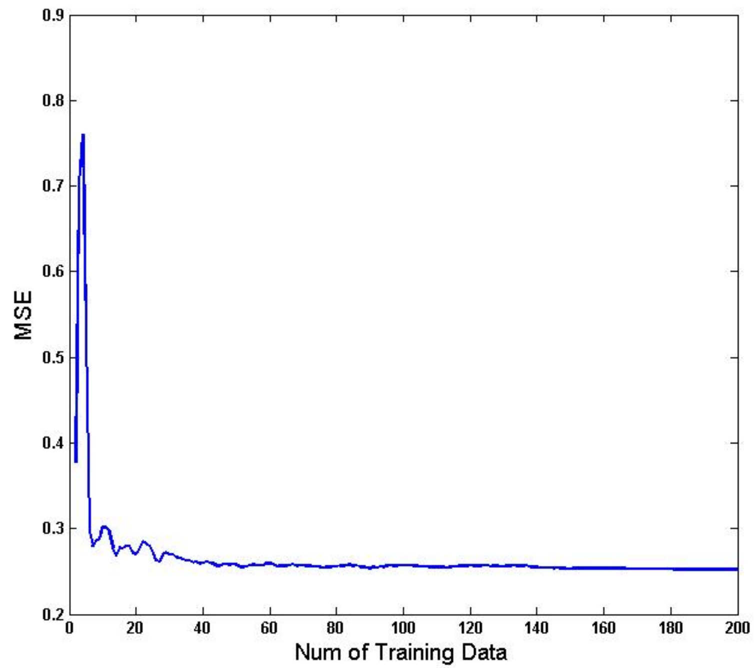


$n = 50$



Linear regression: generalization

- By increasing the number of training examples, will solution be better?
- Why the mean squared error does not decrease more after reaching a level?



Linear regression: types of errors

- <u>Structural error</u>: the error introduced by the limited function class (infinite training data):

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} E_{\boldsymbol{x},y}[(y - \boldsymbol{w}^T\boldsymbol{x})^2]$$

$$\text{Structural error: } E_{\boldsymbol{x},y}\left[\left(y - \boldsymbol{w}^{*T}\boldsymbol{x}\right)^2\right]$$

where $\boldsymbol{w}^* = (w_0^*, \cdots, w_d^*)$ are the optimal linear regression parameters (infinite training data)

Linear regression: types of errors

- <u>Approximation error</u> measures how close we can get to the optimal linear predictions with limited training data:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, E_{\boldsymbol{x},y}[(y - \boldsymbol{w}^T\boldsymbol{x})^2]$$

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2$$

Approximation error: $E_{\boldsymbol{x}}\left[\left({\boldsymbol{w}^*}^T\boldsymbol{x} - \widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)^2\right]$

Where $\widehat{\boldsymbol{w}}$ are the parameter estimates based on a small training set (so themselves are random variables).

Linear regression: error decomposition

- The expected error can decompose into the sum of structural and approximation errors

$$E_{\boldsymbol{x},y}[(y - \widehat{\boldsymbol{w}}^T\boldsymbol{x})^2]$$
$$= E_{\boldsymbol{x},y}\left[\left(y - {\boldsymbol{w}^*}^T\boldsymbol{x}\right)^2\right] + E_{\boldsymbol{x}}\left[\left({\boldsymbol{w}^*}^T\boldsymbol{x} - \widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)^2\right]$$

- Derivation

$$E_{\boldsymbol{x},y}[(y - \widehat{\boldsymbol{w}}^T\boldsymbol{x})^2] = E_{\boldsymbol{x},y}\left[\left(y - {\boldsymbol{w}^*}^T\boldsymbol{x} + {\boldsymbol{w}^*}^T\boldsymbol{x} - \widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)^2\right]$$
$$= E_{\boldsymbol{x},y}\left[\left(y - {\boldsymbol{w}^*}^T\boldsymbol{x}\right)^2\right] + E_{\boldsymbol{x}}\left[\left({\boldsymbol{w}^*}^T\boldsymbol{x} - \widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)^2\right]$$
$$+ 2E_{\boldsymbol{x},y}\left[\left(y - {\boldsymbol{w}^*}^T\boldsymbol{x}\right)\left({\boldsymbol{w}^*}^T\boldsymbol{x} - \widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)\right]$$

Linear regression: error decomposition

- The expected error can decompose into the sum of structural and approximation errors

$$E_{\boldsymbol{x},y}[(y-\widehat{\boldsymbol{w}}^T\boldsymbol{x})^2]$$
$$=E_{\boldsymbol{x},y}\left[\left(y-\boldsymbol{w}^{*T}\boldsymbol{x}\right)^2\right]+E_{\boldsymbol{x}}\left[\left(\boldsymbol{w}^{*T}\boldsymbol{x}-\widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)^2\right]$$

- Derivation

$$E_{\boldsymbol{x},y}[(y-\widehat{\boldsymbol{w}}^T\boldsymbol{x})^2]=E_{\boldsymbol{x},y}\left[\left(y-\boldsymbol{w}^{*T}\boldsymbol{x}+\boldsymbol{w}^{*T}\boldsymbol{x}-\widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)^2\right]$$
$$=E_{\boldsymbol{x},y}\left[\left(y-\boldsymbol{w}^{*T}\boldsymbol{x}\right)^2\right]+E_{\boldsymbol{x}}\left[\left(\boldsymbol{w}^{*T}\boldsymbol{x}-\widehat{\boldsymbol{w}}^T\boldsymbol{x}\right)^2\right]$$
$$+\ 0$$

Note: Optimality condition for $\boldsymbol{w}^*$ give us $E_{\boldsymbol{x},y}\left[\left(y-\boldsymbol{w}^{*T}\boldsymbol{x}\right)\boldsymbol{x}\right]=0$
since $\nabla_{\boldsymbol{w}}E_{\boldsymbol{x},y}[(y-\boldsymbol{w}^T\boldsymbol{x})^2]|_{\boldsymbol{w}*}=0$