Linear regression, Gradient Descent

Iran University of Science and Technology

By: M. S. Tahaei, PhD.

Winter 2025

Outline

- Introduction to Learning
- Linear Regression
- Gradient Descent
- Generalized Linear Regression

A Definition of ML

- Tom Mitchell (1998):Well-posed learning problem
  - “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E”.

- Using the observed data to make better decisions
  - Generalizing from the observed data

ML Definition: Example

- Consider an email program that learns how to filter spam according to emails you do or do not mark as spam.
  - T: Classifying emails as spam or not spam.
  - E: Watching you label emails as spam or not spam.
  - P: The number (or fraction) of emails correctly classified as spam/not spam.

The essence of machine learning

- A pattern exist
- We do not know it mathematically
- We have data on it

Example: Home Price

- Housing price prediction

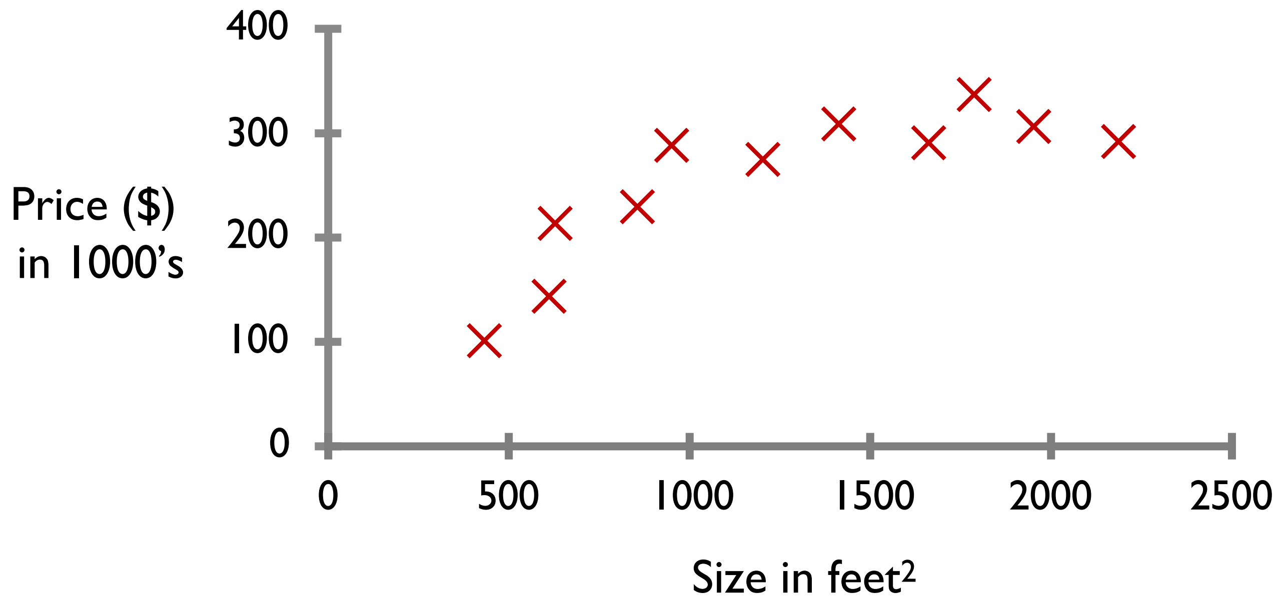


Figure adopted from slides of Andrew Ng, Machine Learning course, Stanford.

Example: Bank loan

- Applicant form as the input:

age	23 years
gender	male
annual salary	\$30,000
years in residence	1 year
years in job	1 year
current debt	\$15,000
...	...

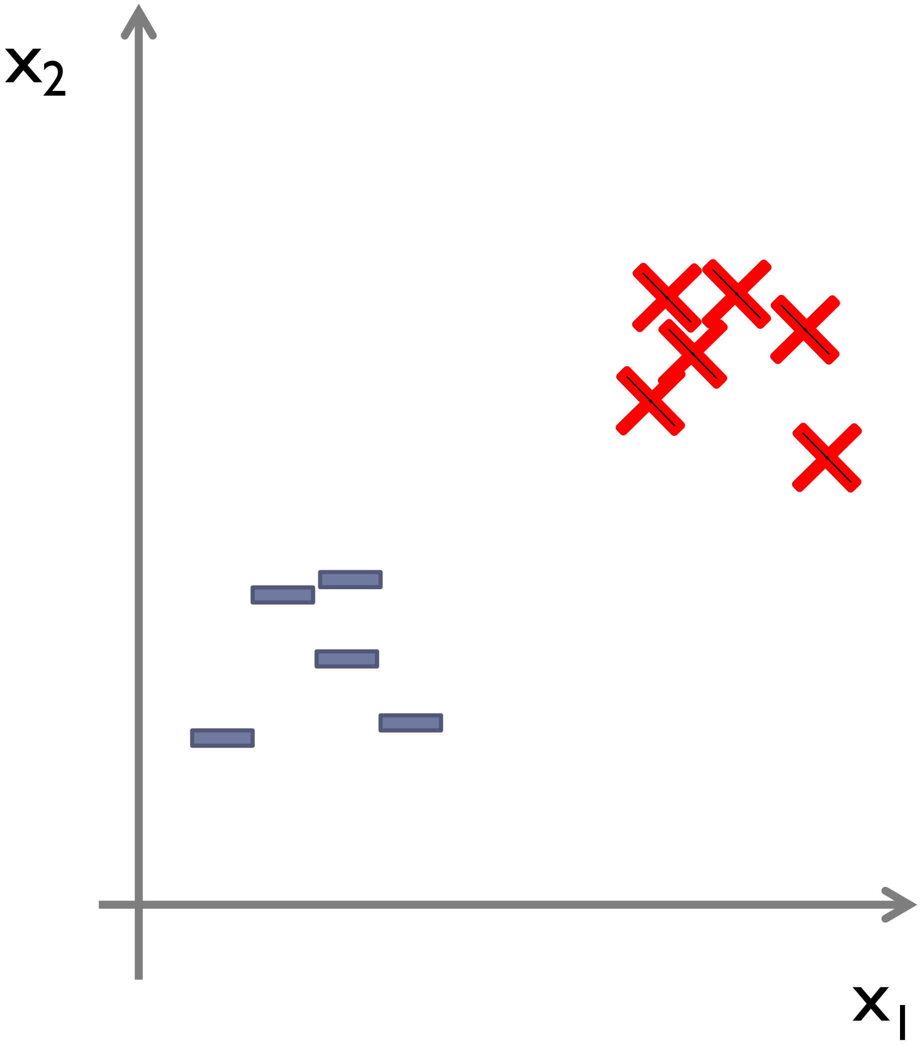
- Output: approving or denying the request

Components of (Supervised) Learning

- Unknown target function: $f: \mathcal{X} \to \mathcal{Y}$
  - Input space: $\mathcal{X}$
  - Output space: $\mathcal{Y}$
- Training data: $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$

- Picka formula $g: \mathcal{X} \to \mathcal{Y}$ that approximates the target function f
  - selected from a set of hypotheses $\mathcal{H}$

Training data: Example



Training data

x_1	x_2	y
0.9	2.3	1
3.5	2.6	1
2.6	3.3	1
2.7	4.1	1
1.8	3.9	1
6.5	6.8	-1
7.2	7.5	-1
7.9	8.3	-1
6.9	8.3	-1
8.8	7.9	-1
9.1	6.2	-1

Solution Components

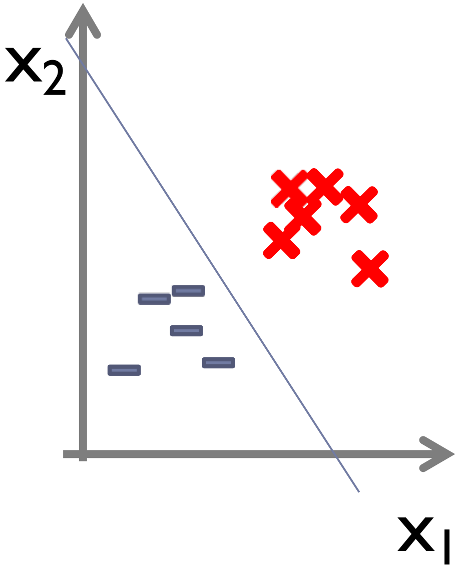
- Learning model composed of:
  - Learning algorithm
  - Hypothesis set

- Perceptron example

Perceptron classifier

- Input $\boldsymbol{x} = [x_1, \ldots, x_d]$

- Classifier:
  - If $\sum_{i=1}^{d} w_i x_i > \text{threshold}$ then output 1
  - else output -1



- The linear formula $g \in \mathcal{H}$ can be written:

$$g(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{d} w_i x_i + w_0 \right)$$

If we add a coordinate $x_0 = 1$ to the input:

$$g(\boldsymbol{x}) = \text{sign}\left(\sum_{i=0}^{d} w_i x_i \right)$$

Vector form

$$g(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$$

Perceptron learning algorithm: linearly separable data

- Give the training data $\left(\boldsymbol{x}^{(1)}, y^{(1)}\right), \ldots, \left(\boldsymbol{x}^{(N)}, y^{(N)}\right)$

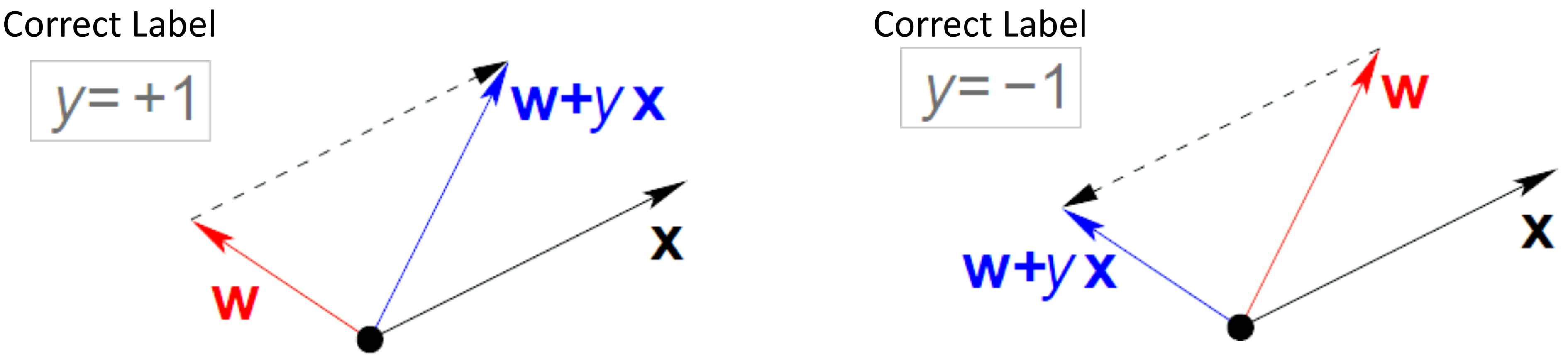
- Misclassified data $\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)$:
$$\operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x}^{(n)}) \neq y^{(n)}$$

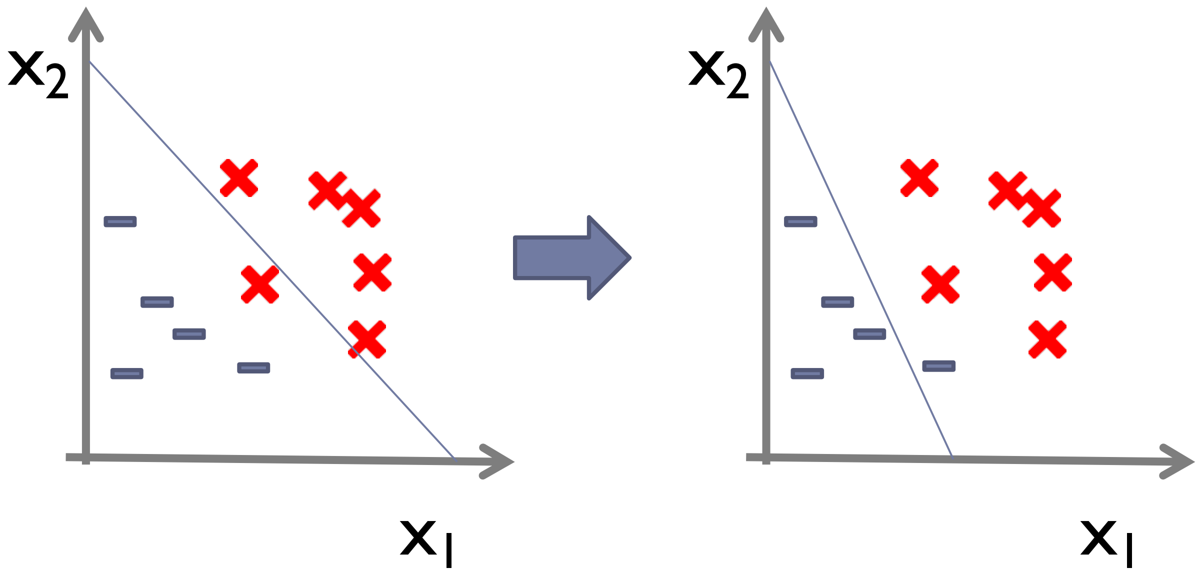
Repeat

Pick a misclassified data $\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)$ from training data and update $\boldsymbol{w}$:
$$\boldsymbol{w} = \boldsymbol{w} + y^{(n)} \boldsymbol{x}^{(n)}$$

Until all training data points are correctly classified by g

Perceptron learning algorithm: Example of weight update





Experience (E) in ML

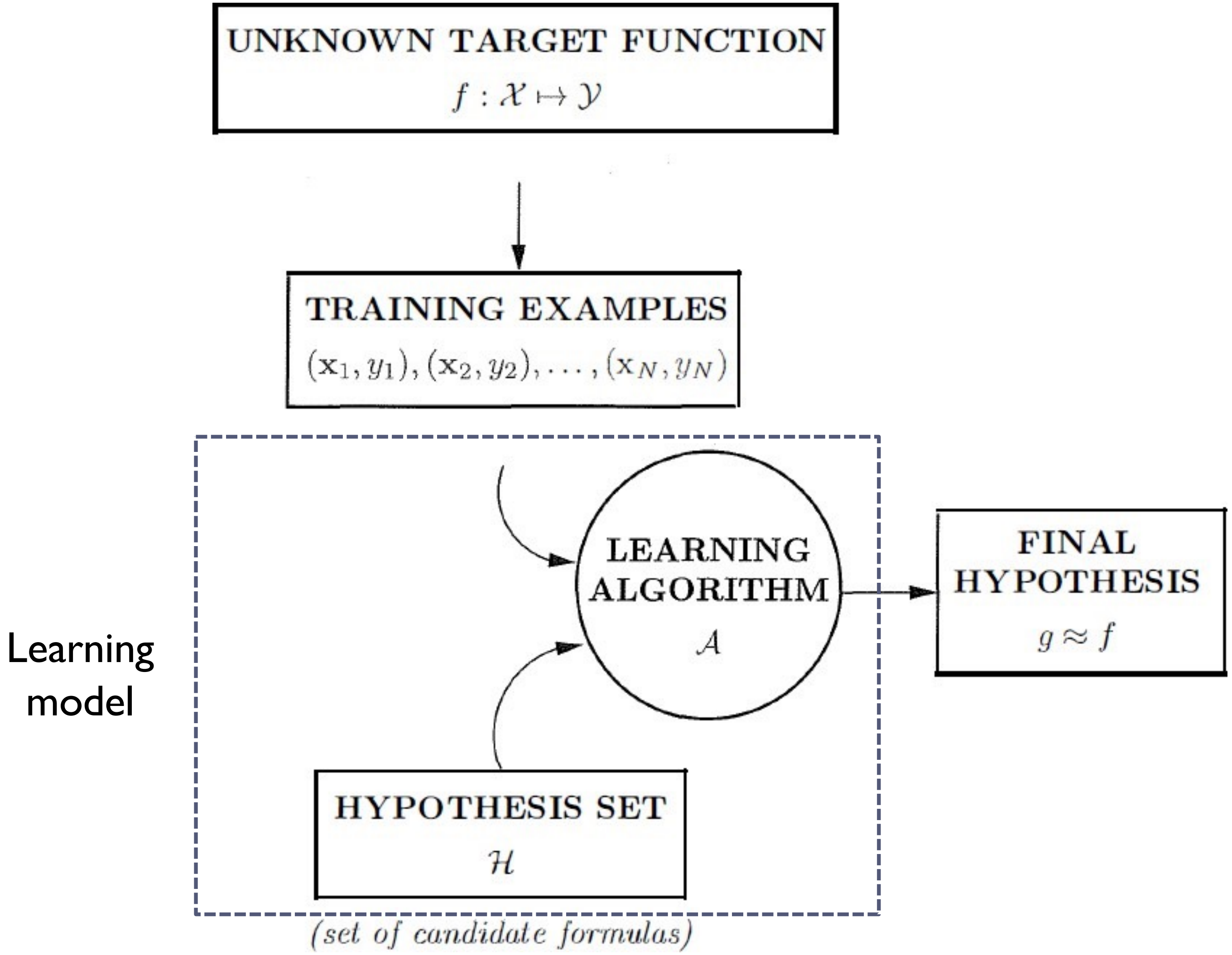
- Basic premise of learning:
  - "Using a set of observations to uncover an underlying process"

- We have different types of (getting) observations in different types or paradigms of ML methods

Main Steps of Learning Tasks

- Selection of hypothesis set (or model specification)
  - Which class of models (mappings) should we use for our data?
- Learning: find mapping f (from hypothesis set) based on the training data
  - Which notion of error should we use? (loss functions)
  - Optimization of loss function to find mapping f
- Evaluation: how well f generalizes to yet unseen examples
  - How do we ensure that the error on future data is minimized? (generalization)

Components of (Supervised) Learning



Linear regression, Cost Function and Gradient Descent

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

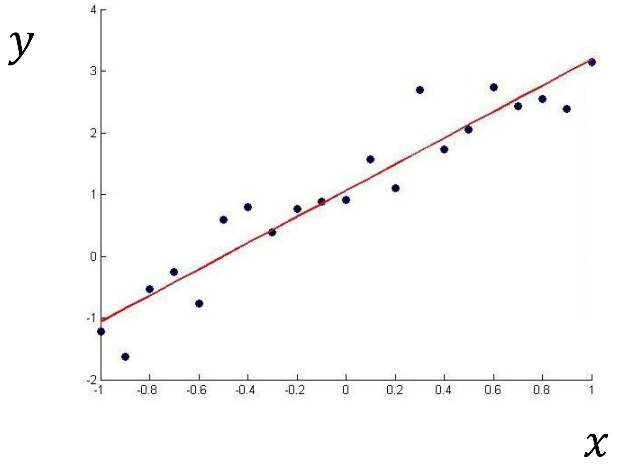
- **Learning (estimation)**: optimization of a cost function
  - Based on the training set $D = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$ and a cost function we find (an estimate) $f \in F$ of the target function

- **Evaluation**: we measure how well f generalizes to unseen examples

Linear regression: hypothesis space

- Univariate

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



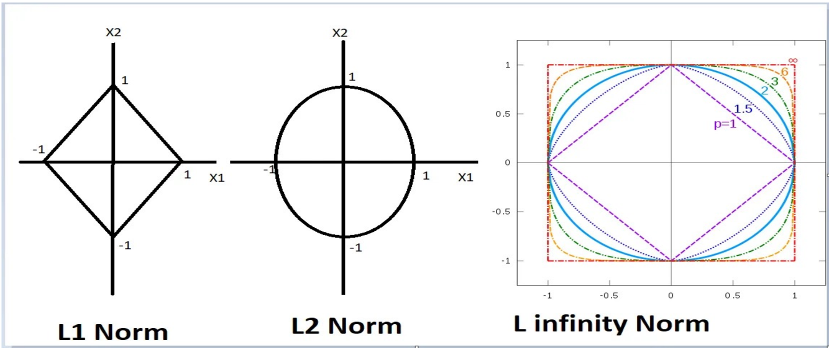
- Multivariate

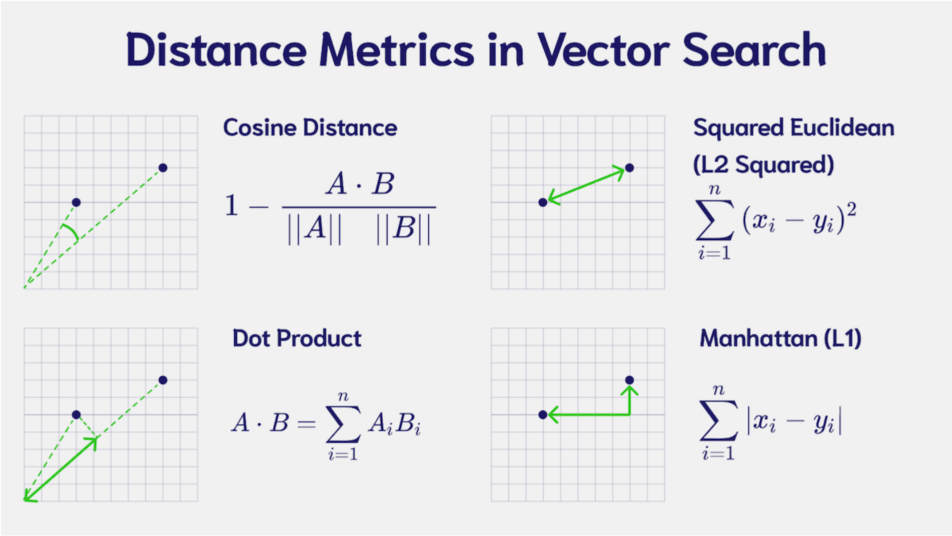
$$f : \mathbb{R}^d \to \mathbb{R} \quad f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x_1 + \ldots w_d x_d$$

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

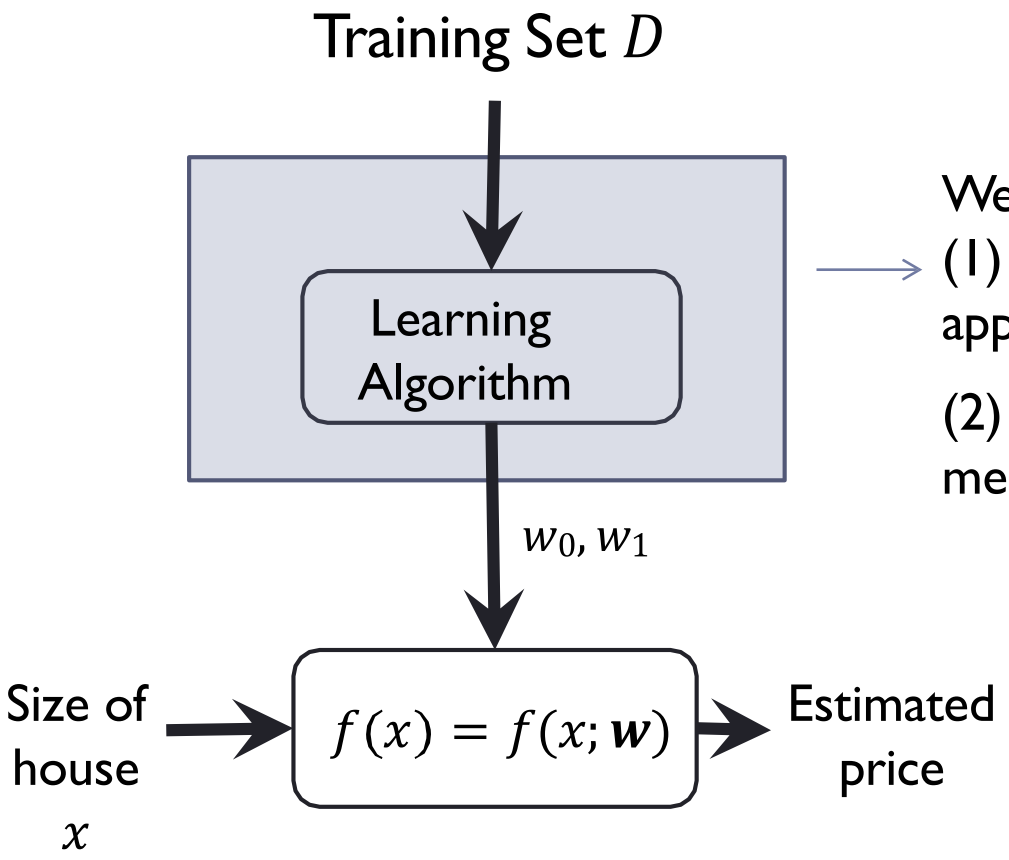
Learning algorithm and distance metrics

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





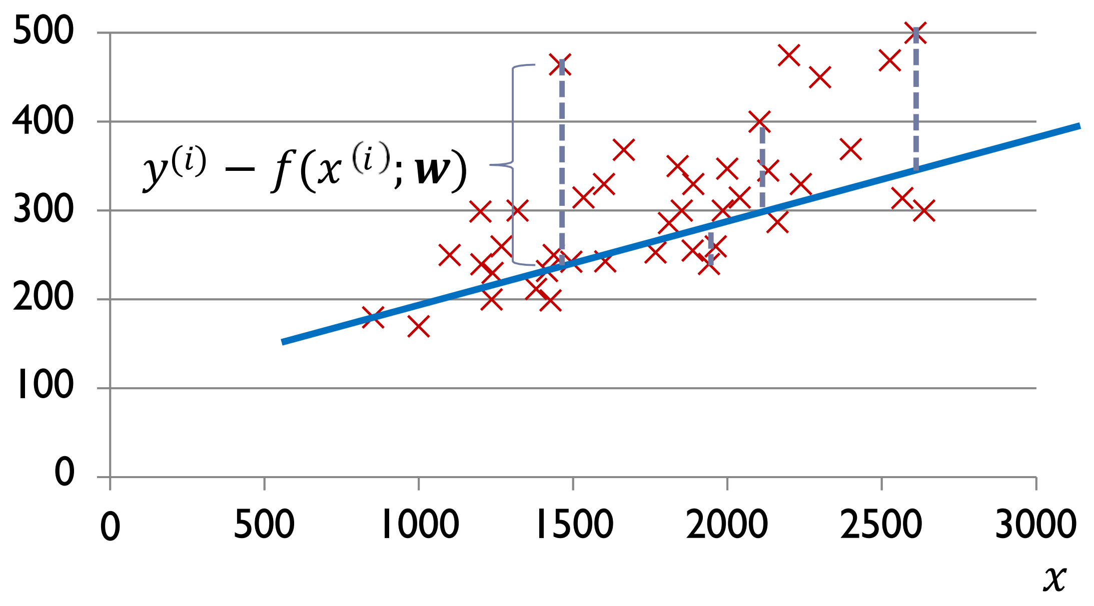
Learning algorithm



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

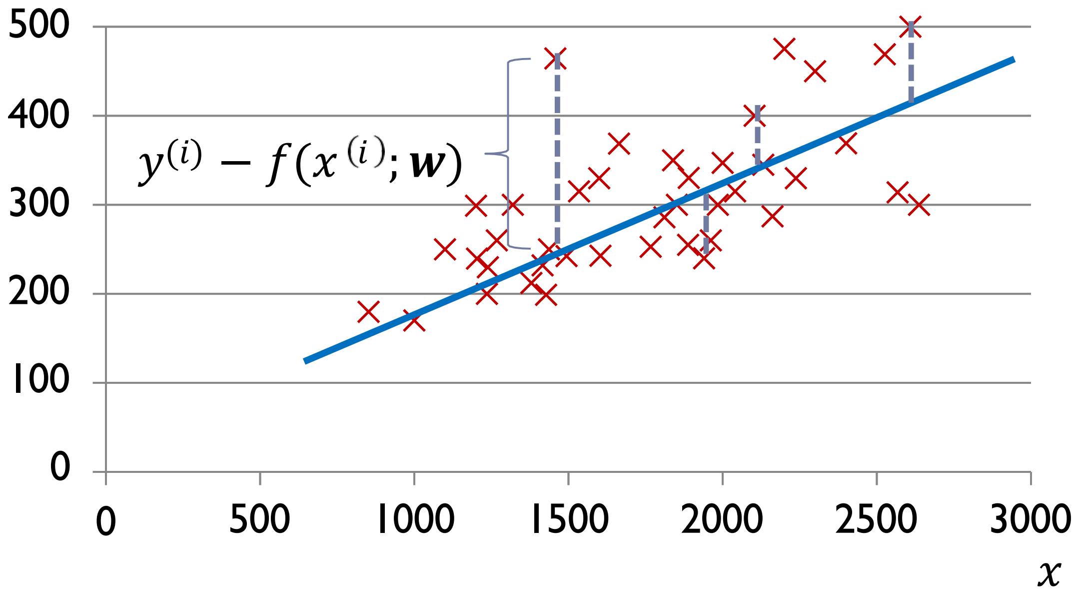
(2) choose $\boldsymbol{w}$ to minimize the error measure

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

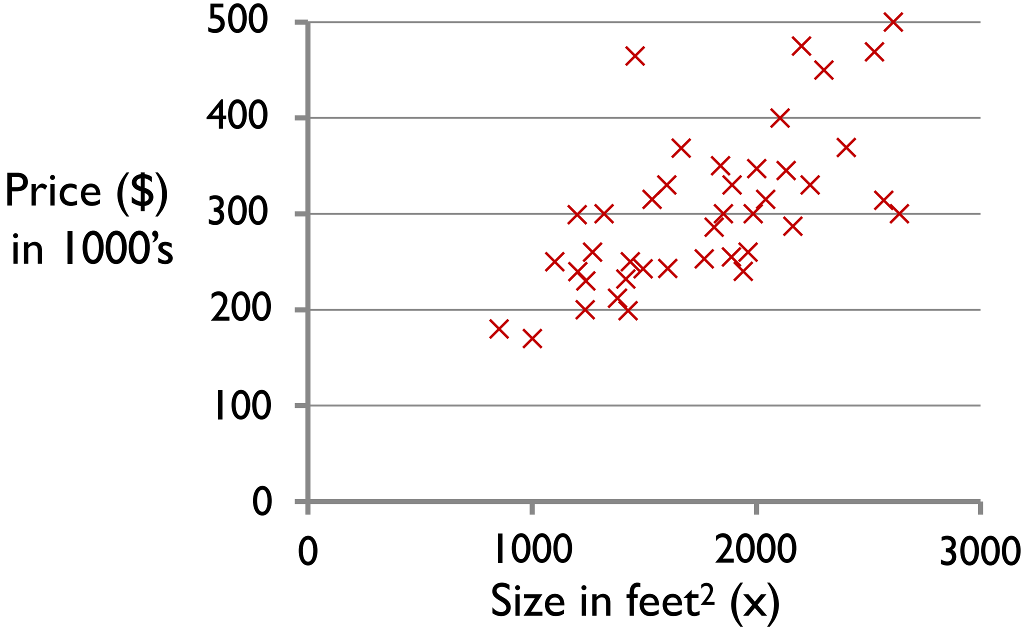
$$J(\boldsymbol{w}) = \sum_{1=i}^{n} \left(y^{(i)} - f(\boldsymbol{x}^{(i)}; \boldsymbol{w})\right)^2$$

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

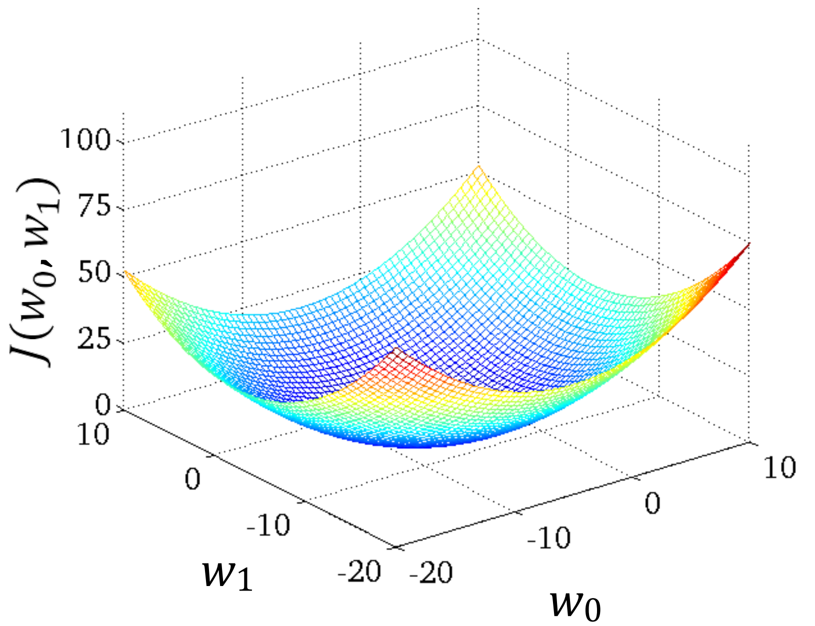
- We want to find the best regression function $f\left(\boldsymbol{x}^{(i)}; \boldsymbol{w}\right)$
  - equivalently, the best $\boldsymbol{w}$

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

Cost function: univariate example



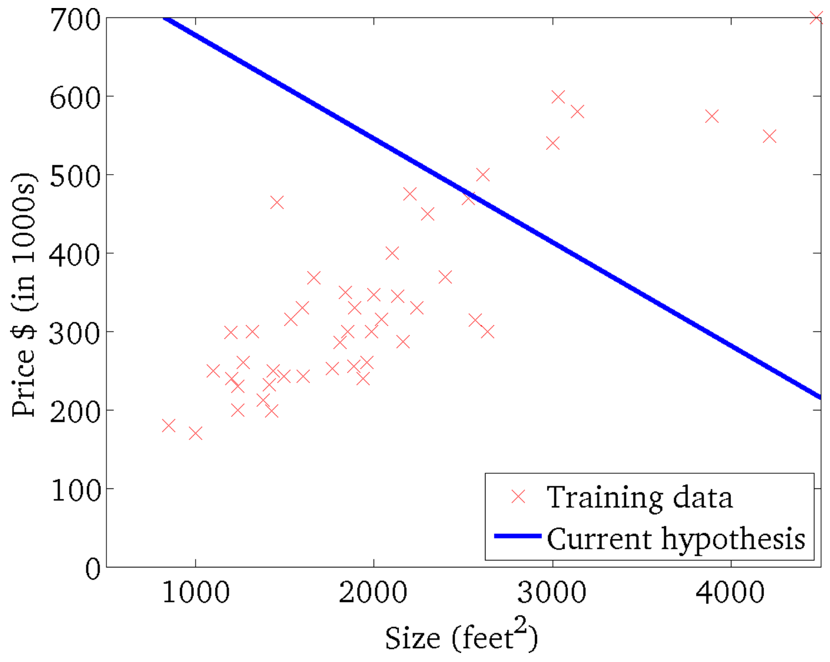
$J(\boldsymbol{w})$
(function of the parameters w_0, w_1)



Cost function: univariate example

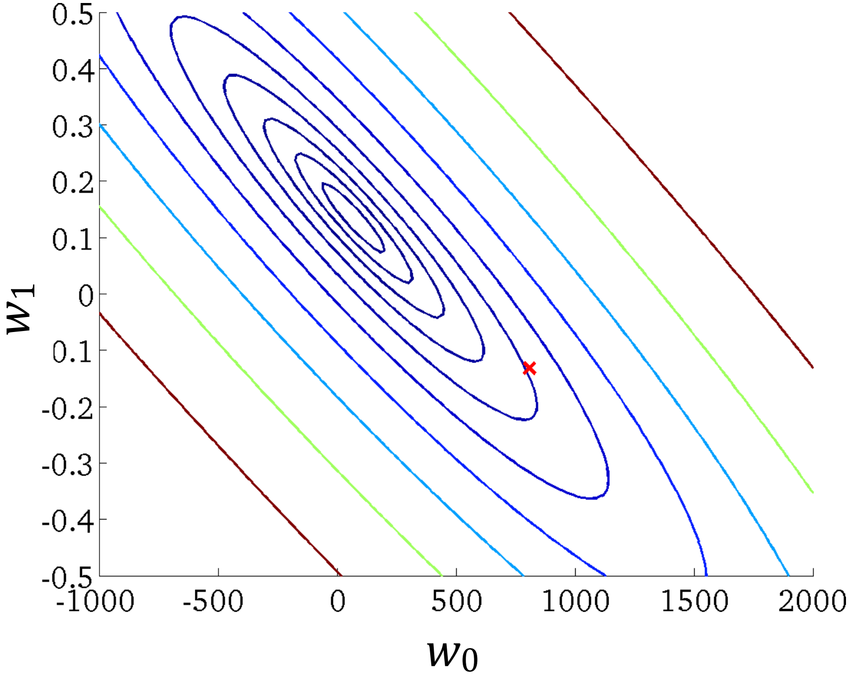
$f(x; w_0, w_1) = w_0 + w_1 x$

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



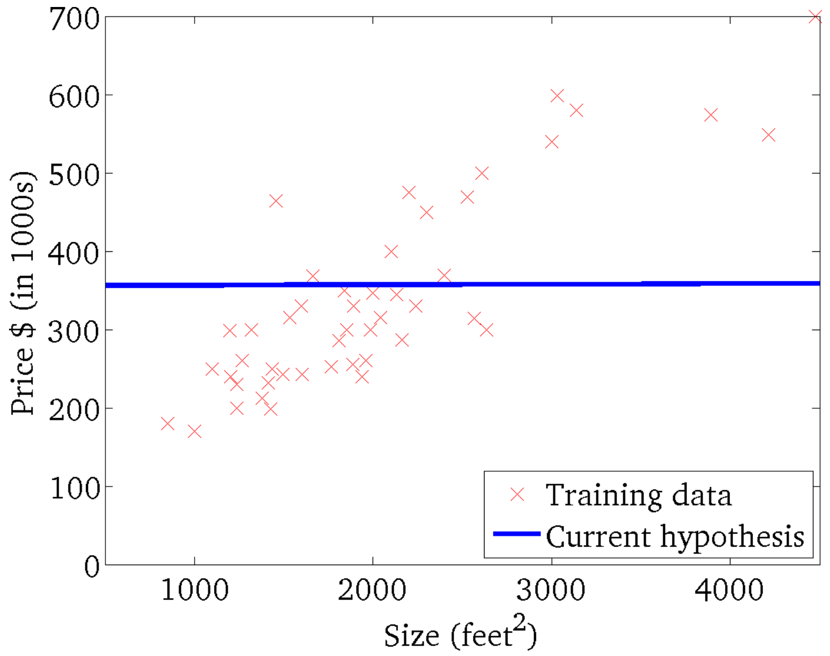
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



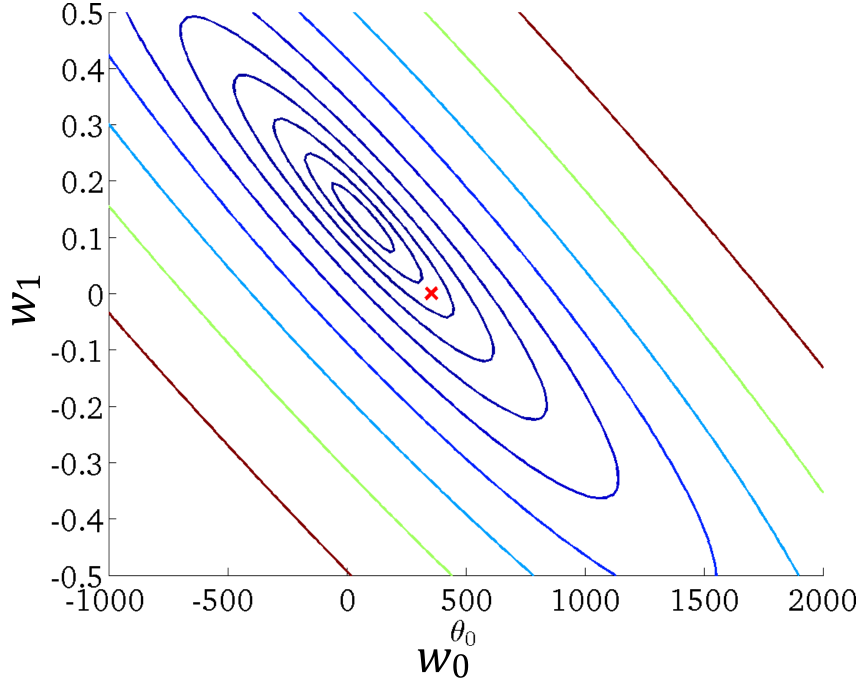
Cost function: univariate example

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



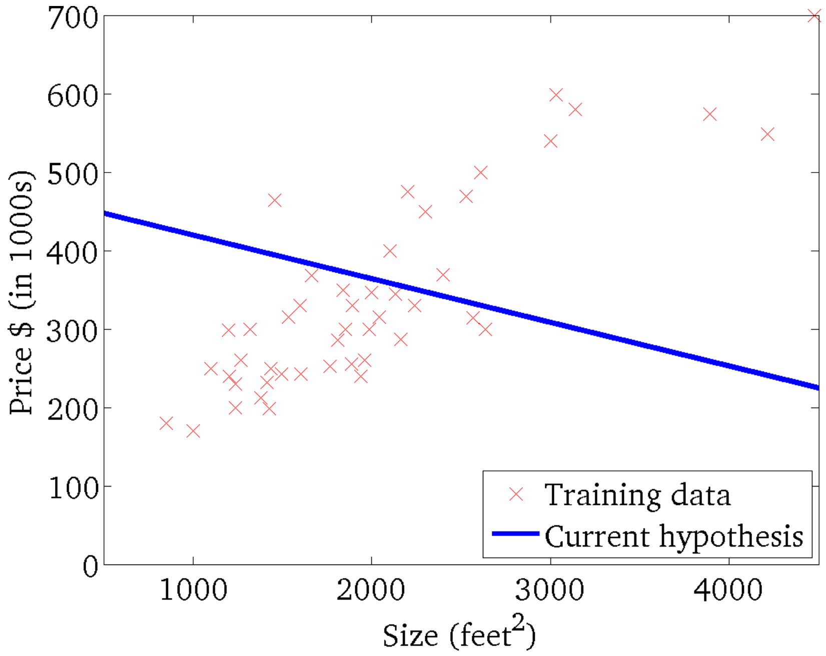
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



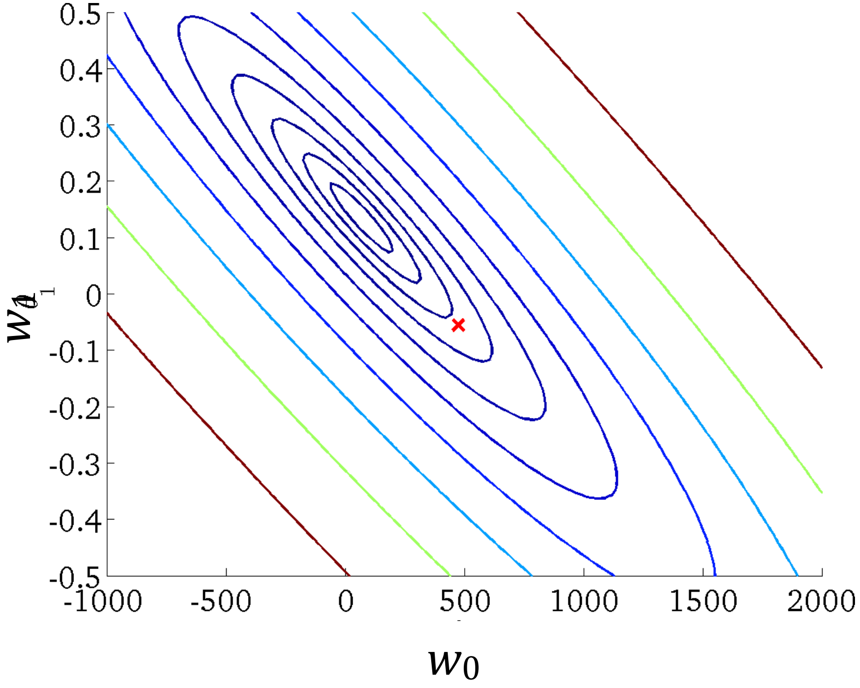
Cost function: univariate example

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



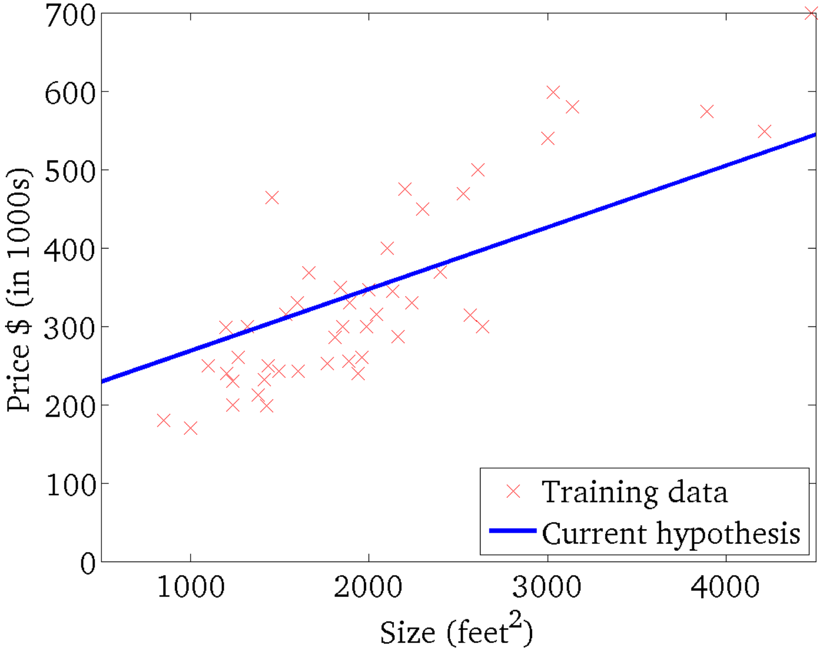
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



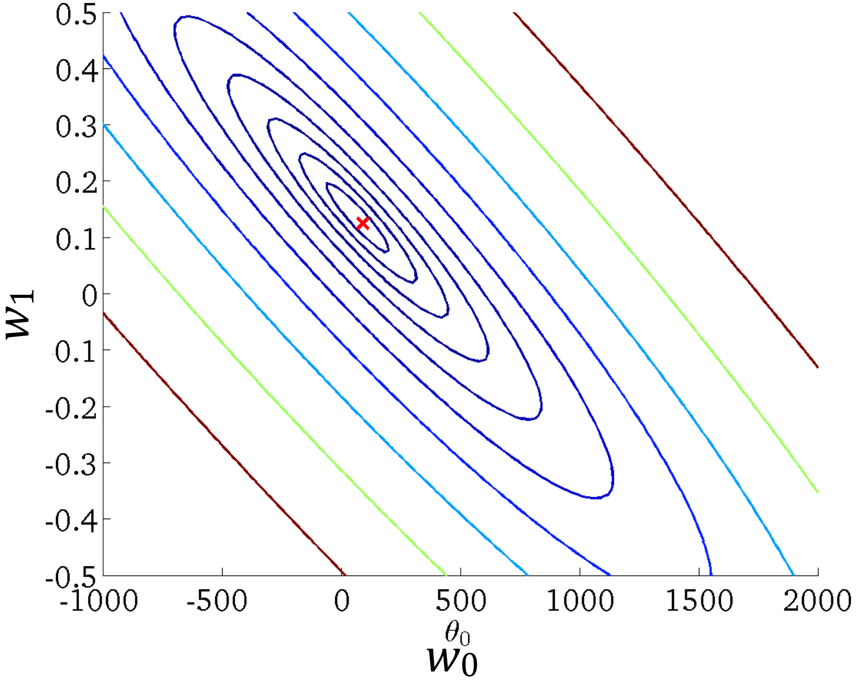
Cost function: univariate example

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



$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)



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

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

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

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

w_1 can be obtained by setting the partial derivative of the SSE to 0 and solving for w_1, ultimately resulting in:

Exercise!

$$\boxed{w_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}} \qquad \boxed{w_0 = \bar{y} - w_1 \bar{x}}$$

Cost function: multivariate for various features

- We have to minimize the empirical squared loss:

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

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

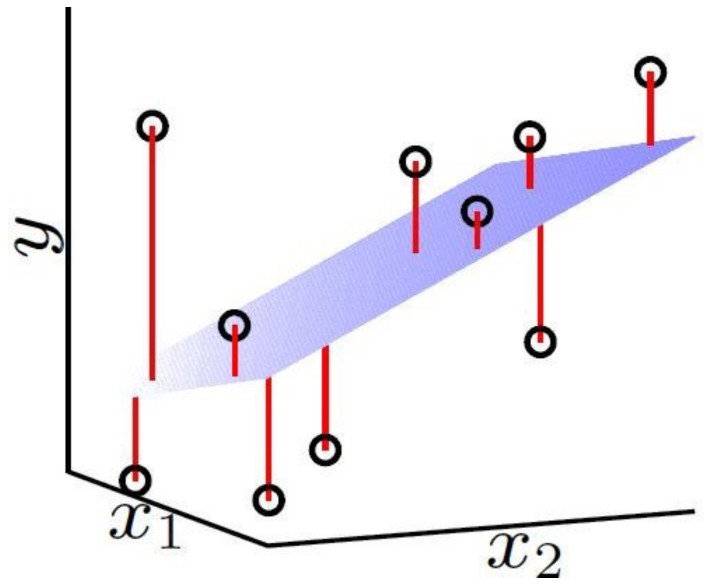
$$\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^T$$

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

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
...	...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0	INLAND
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0	INLAND
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0	INLAND
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0	INLAND
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0	INLAND

20640 rows × 10 columns

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):

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

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

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

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

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

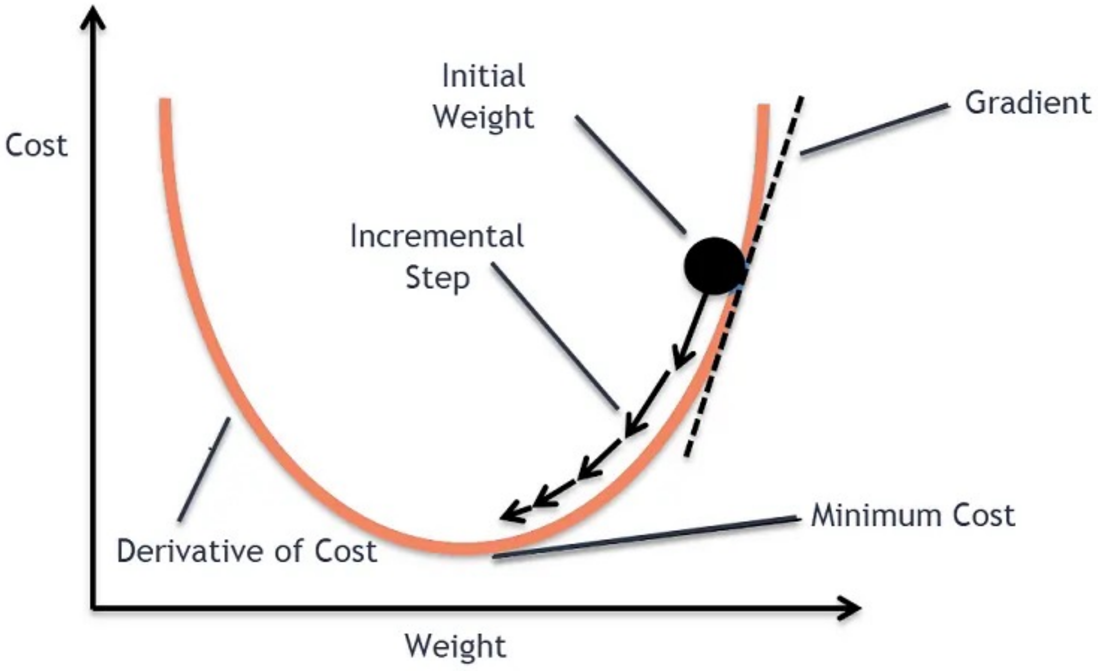
$$\boldsymbol{X}^{\dagger} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$$

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

Another approach for optimizing the sum squared error

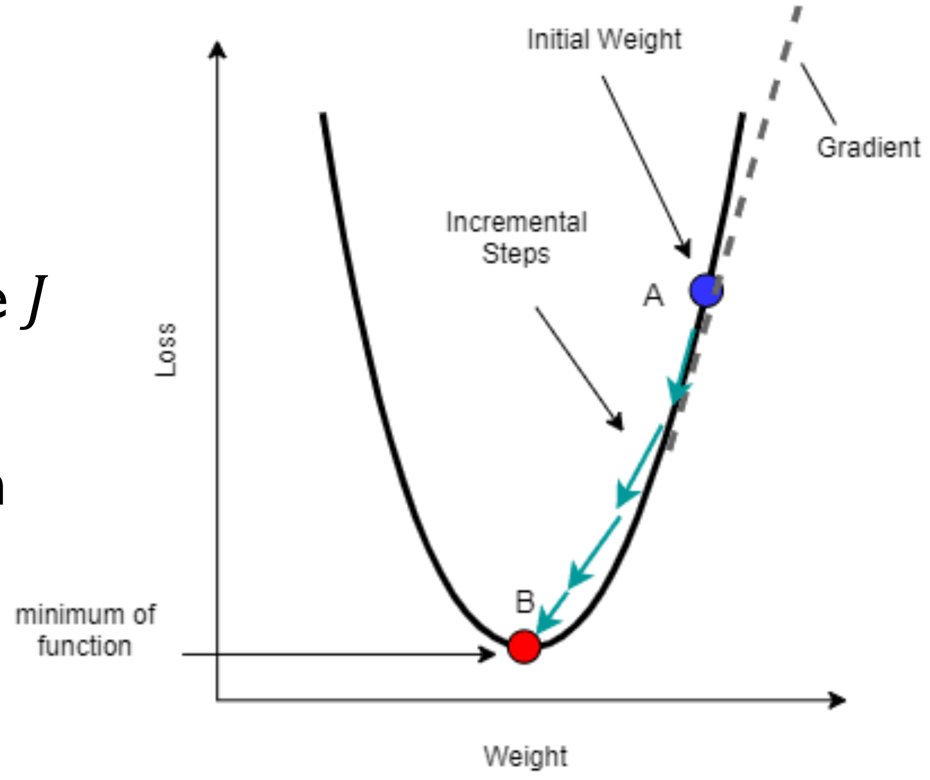
- Iterative approach for solving the following optimization problem:

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



Gradient descent

- Cost function: $J(\boldsymbol{w})$
- Optimization problem: $\boldsymbol{w} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, J(\boldsymbol{w})$
- 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



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 $\boldsymbol{w}^t$:

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

  - $J(\boldsymbol{w})$ <u>decreases fastest</u> 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

Gradient descent

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

Step size
(Learning rate parameter)

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

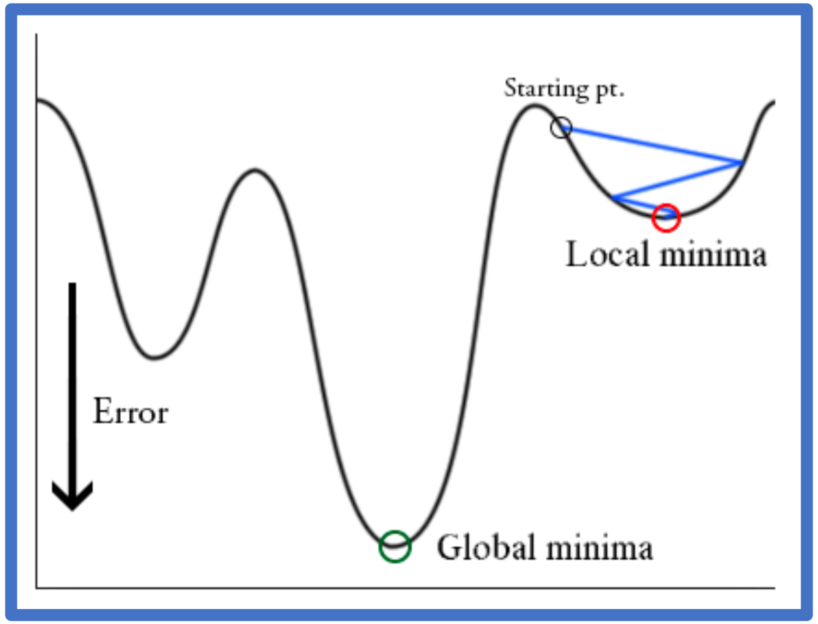
$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = [\frac{\partial J(\boldsymbol{w})}{\partial w_1}, \frac{\partial J(\boldsymbol{w})}{\partial w_2}, \dots, \frac{\partial J(\boldsymbol{w})}{\partial w_d}]$$

- 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.

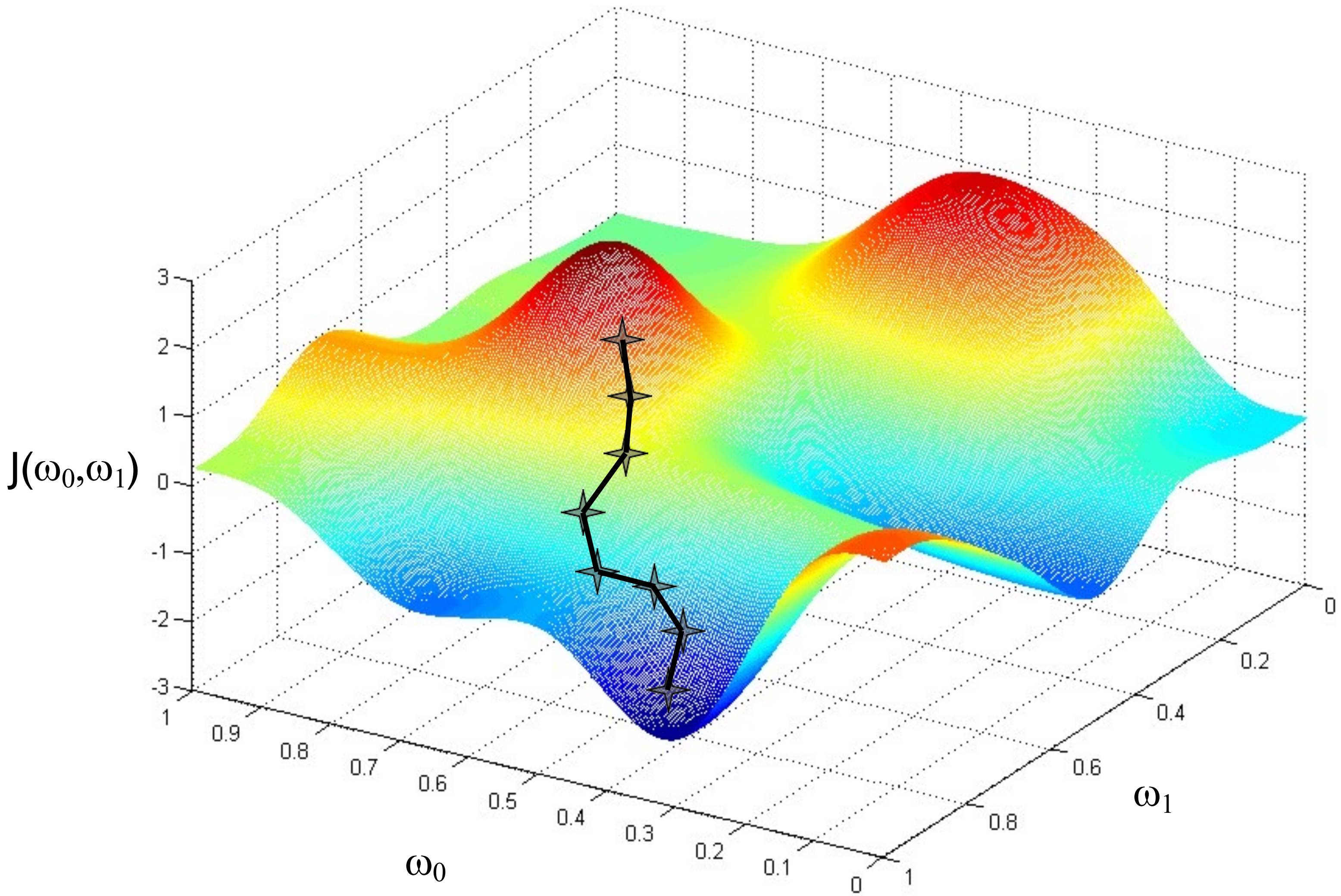
Gradient descent

- Local minima problem

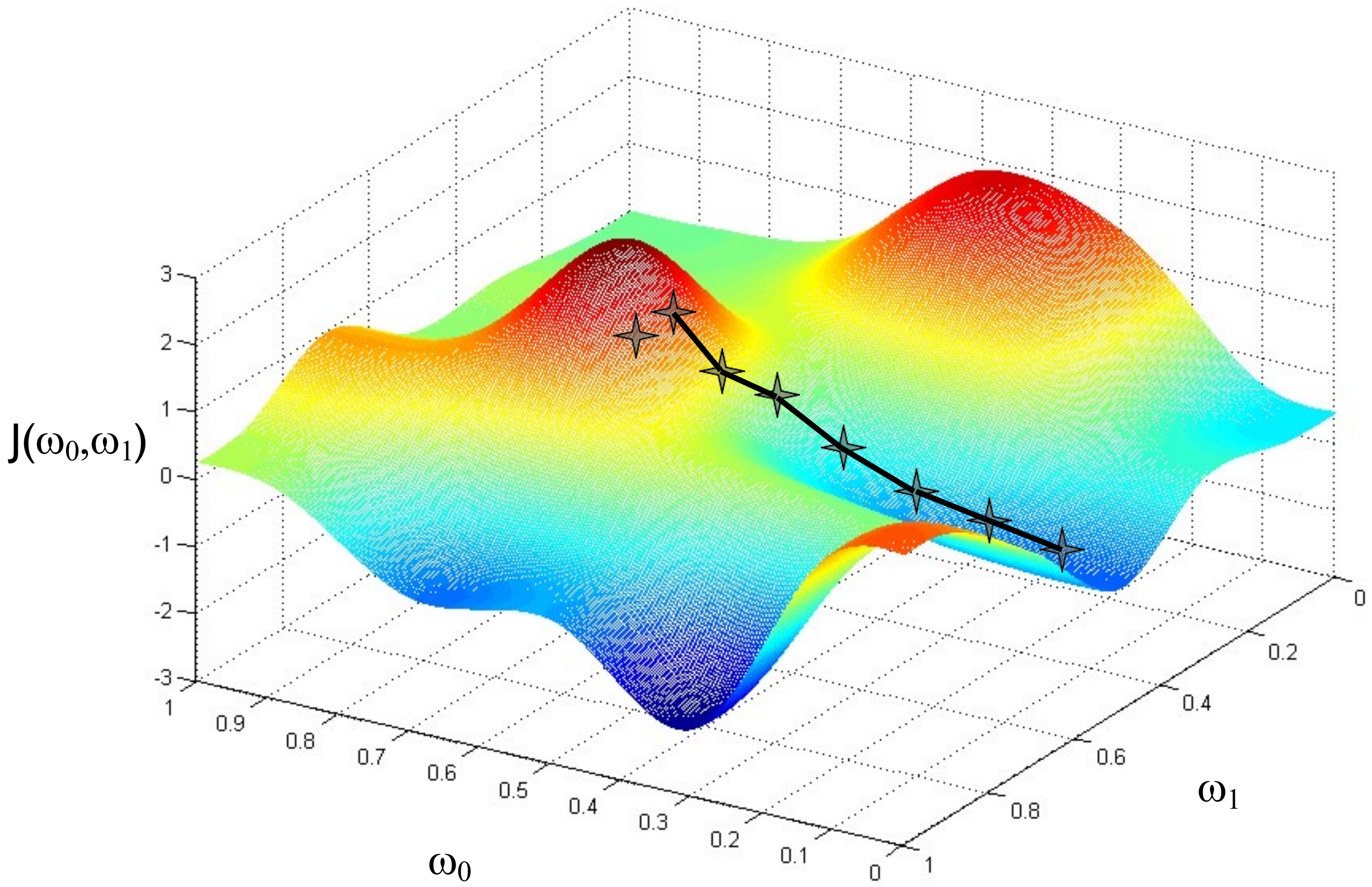
- However, when J is convex, all local minima are also global minima $\Rightarrow$ gradient descent can converge to the global solution.



Problem of gradient descent with non-convex cost functions



Problem of gradient descent with non-convex cost functions



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

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

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

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

Gradient descent for SSE cost function

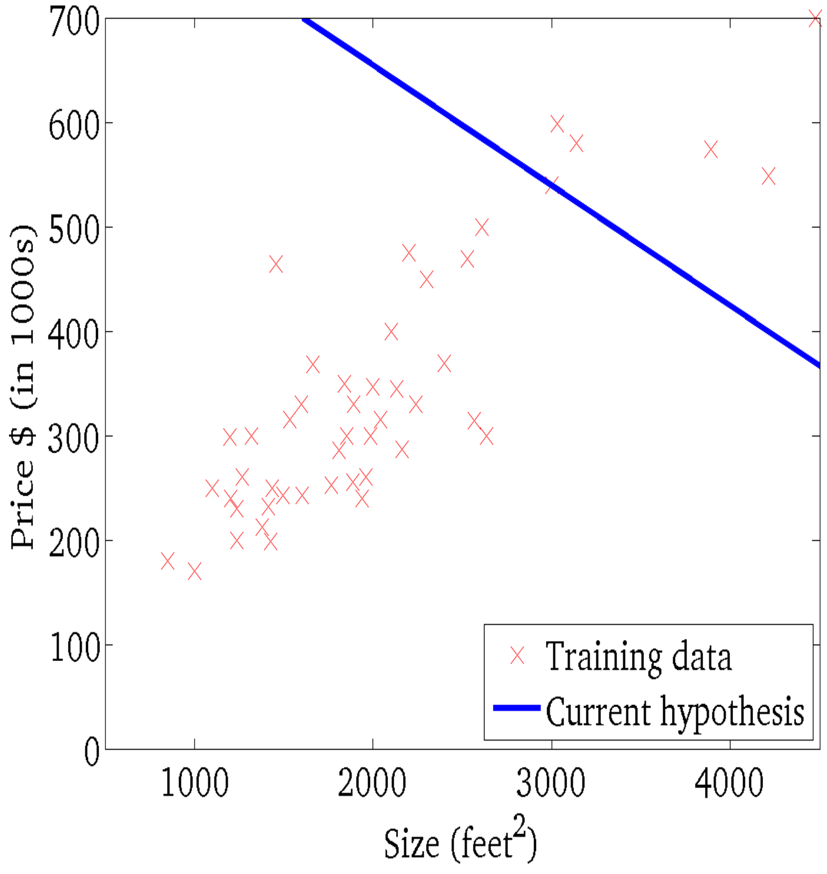
- Weight update rule: $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$

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

Batch mode: each step considers all training data

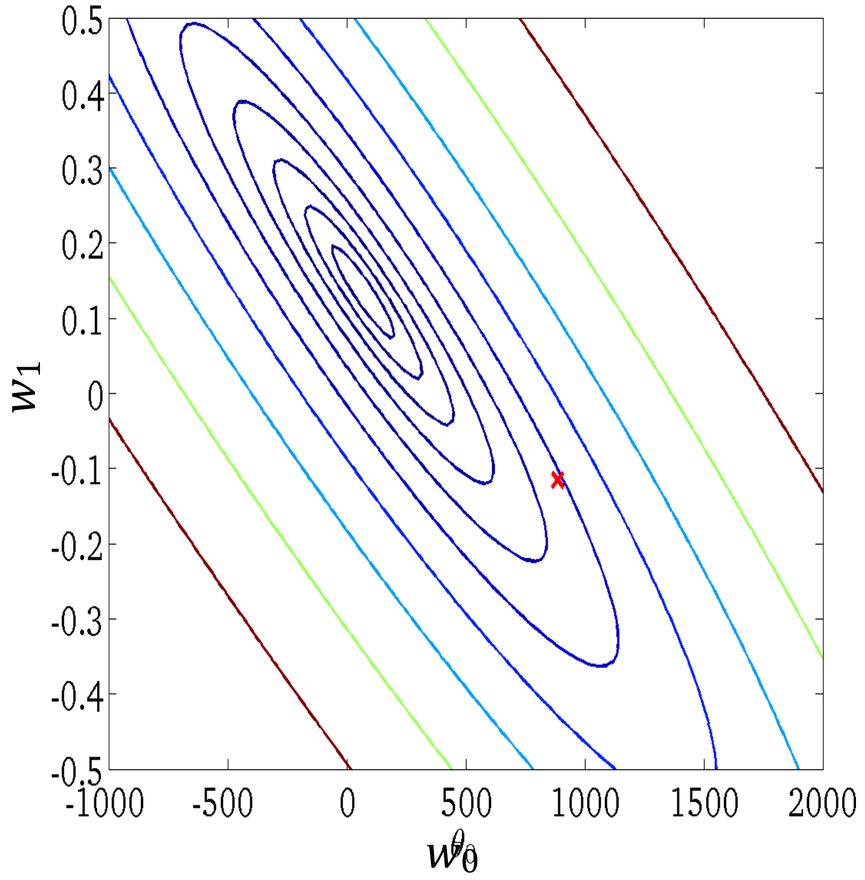
- η: too small → gradient descent can be slow.
- η: too large → gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

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



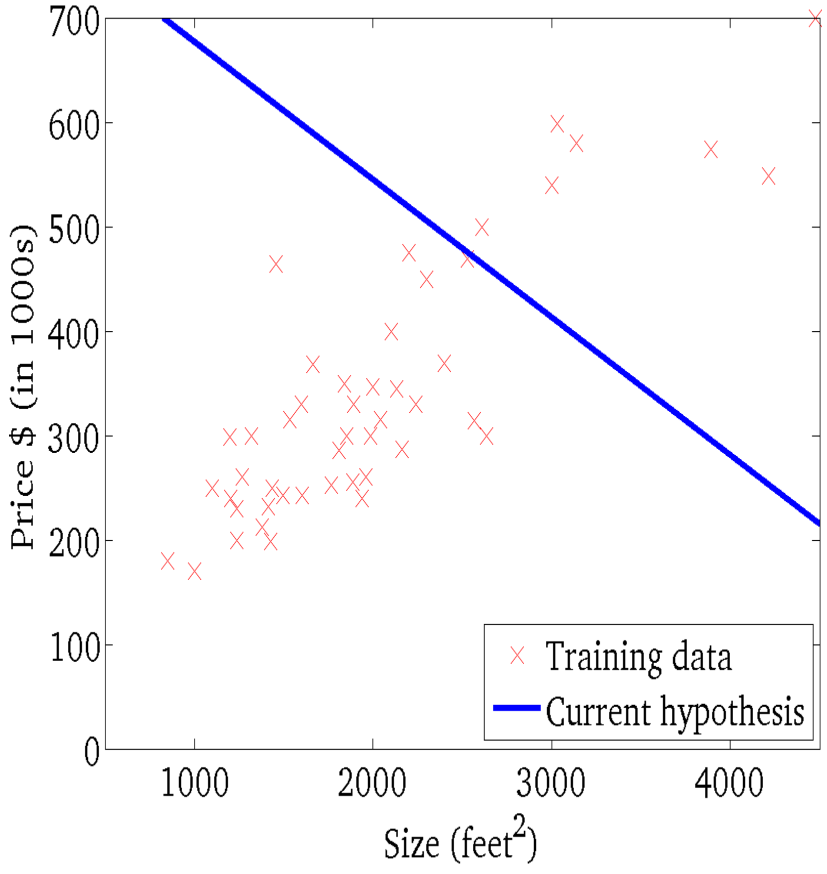
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



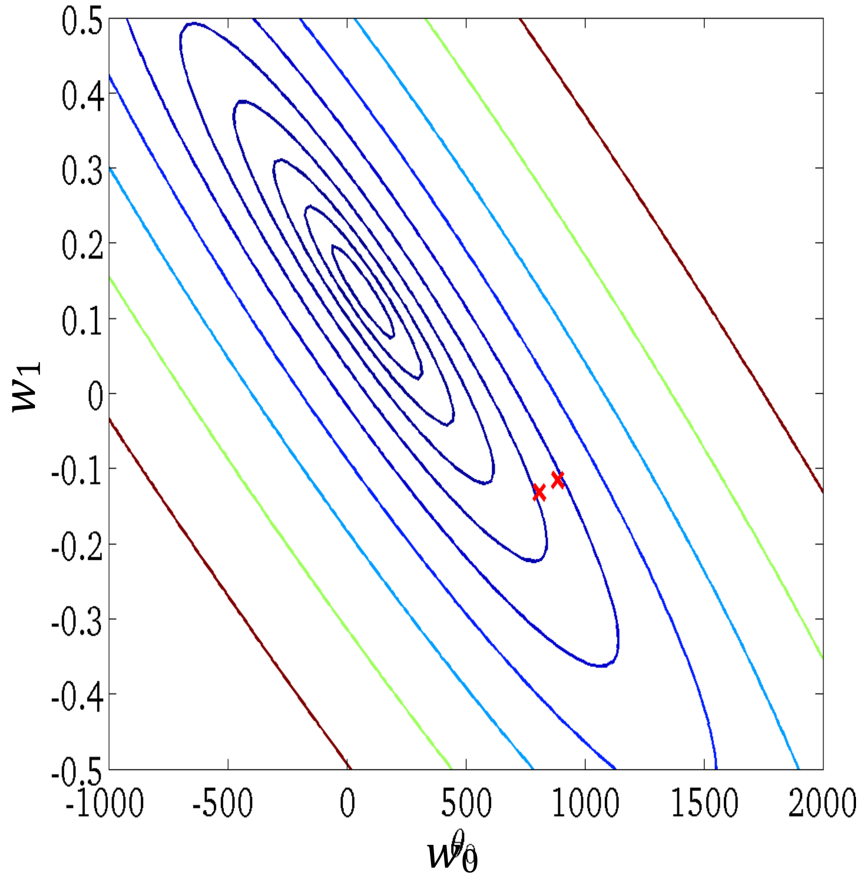
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



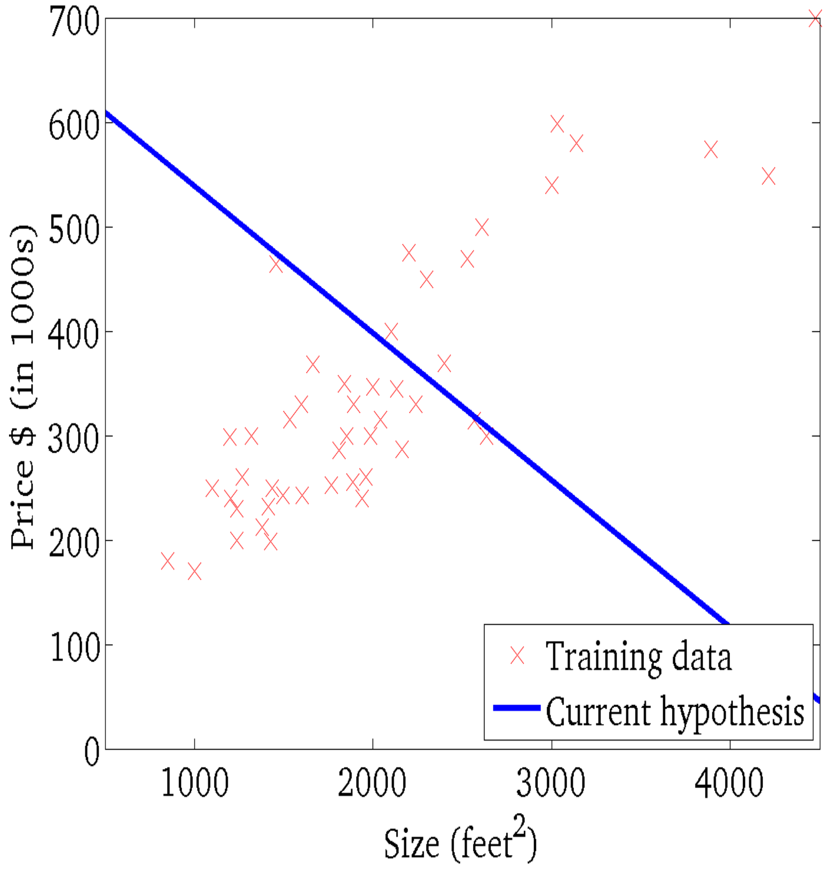
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



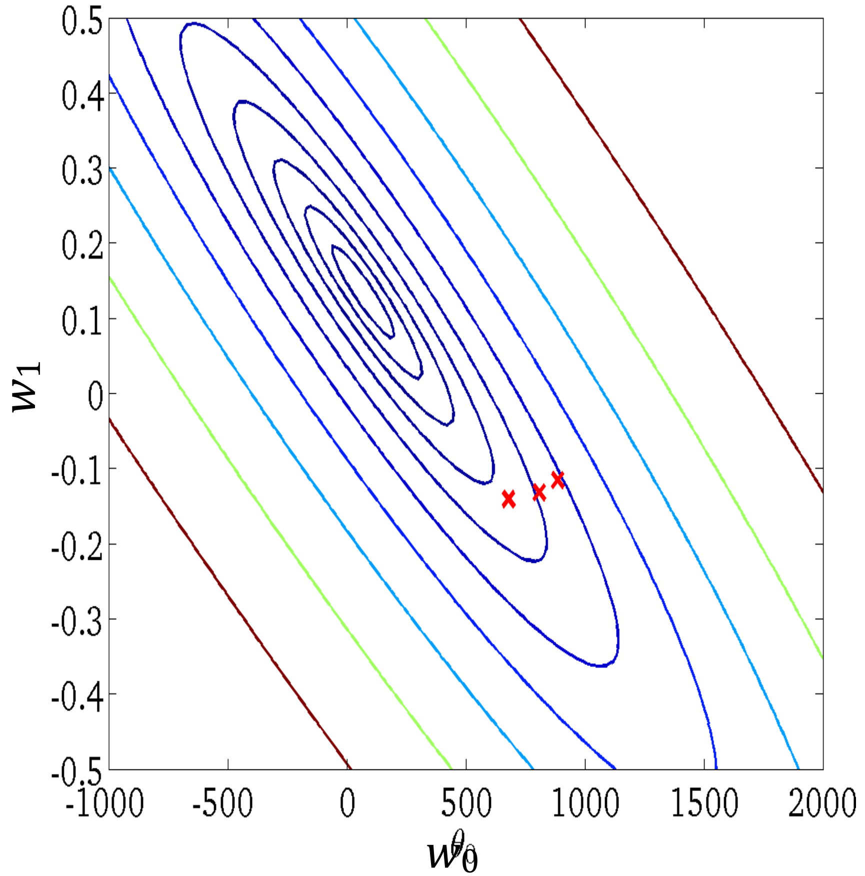
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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

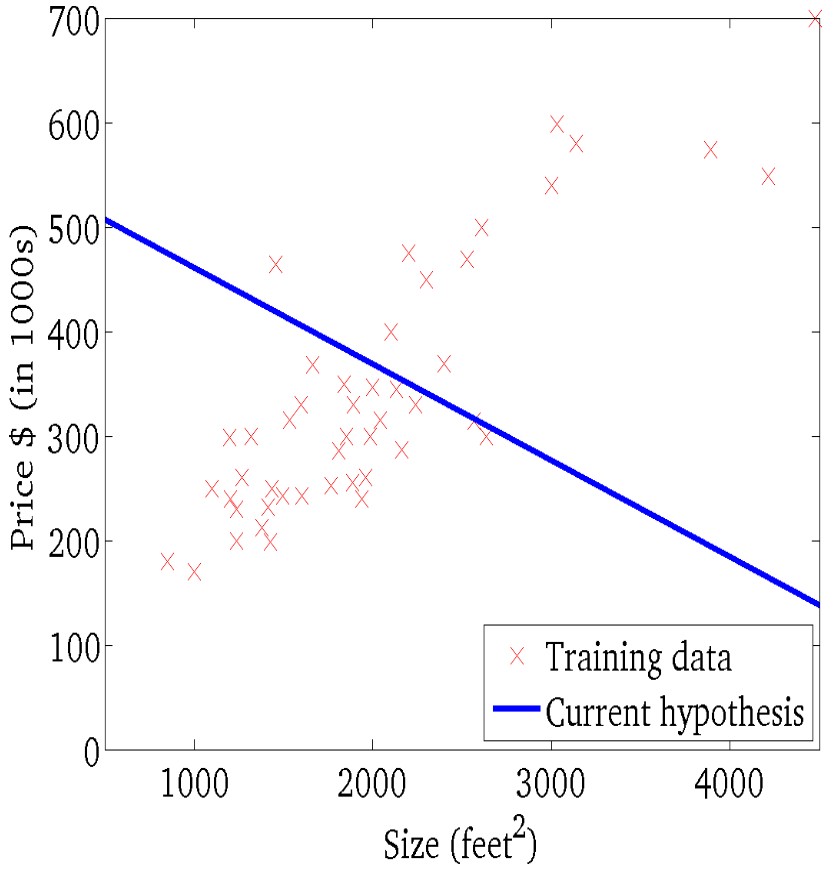


$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)

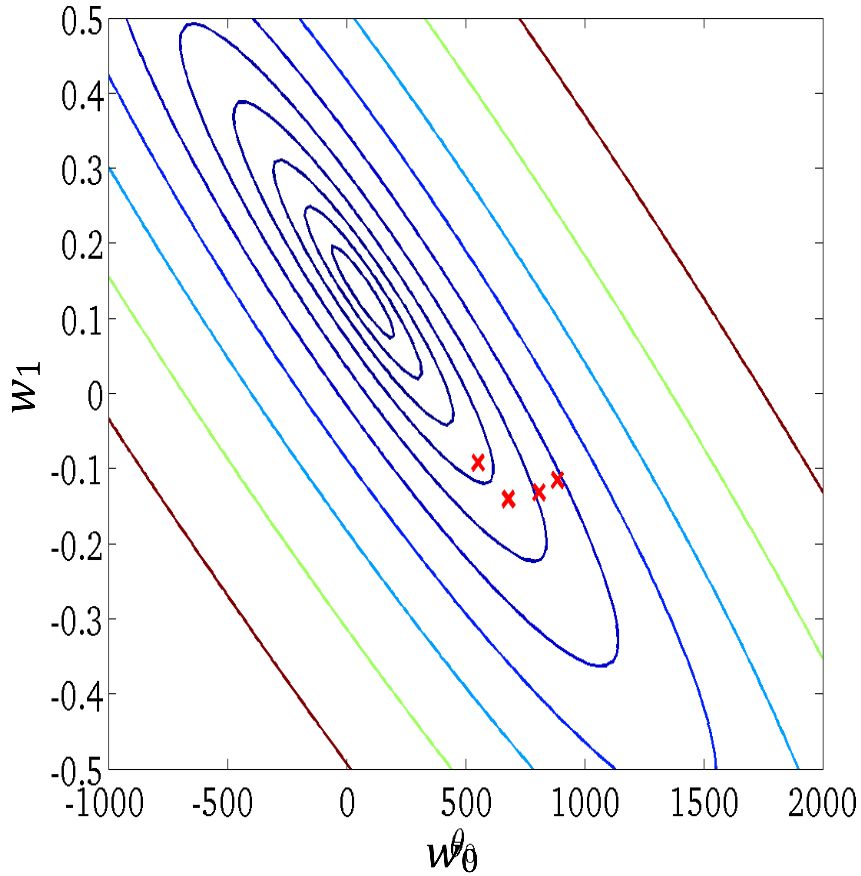


$f(x; w_0, w_1) = w_0 + w_1 x$



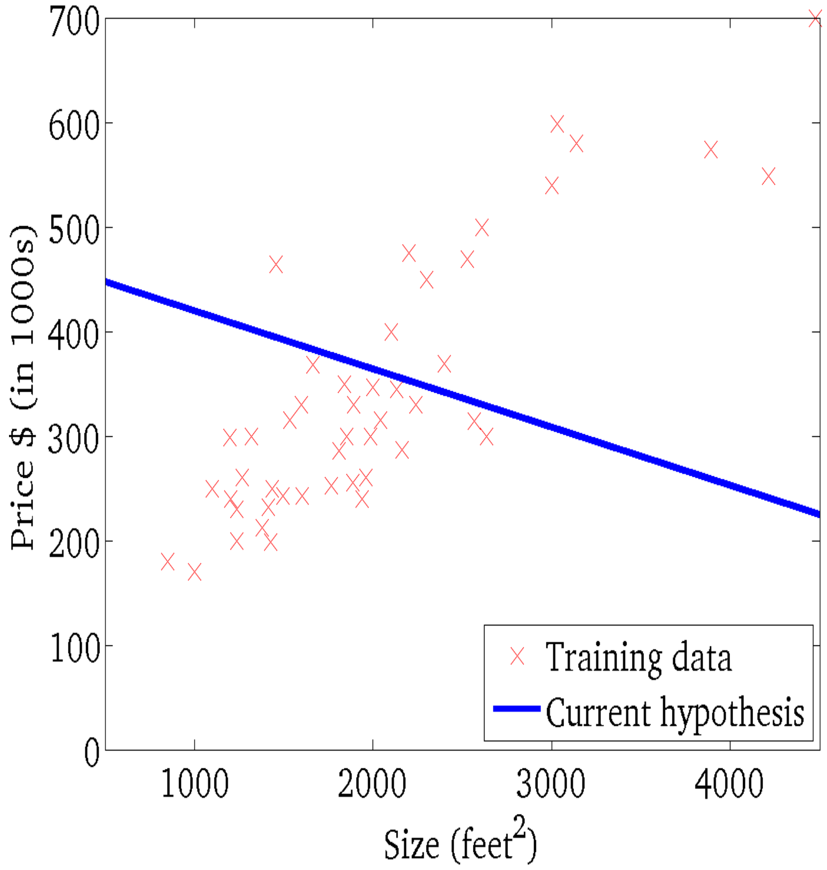
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



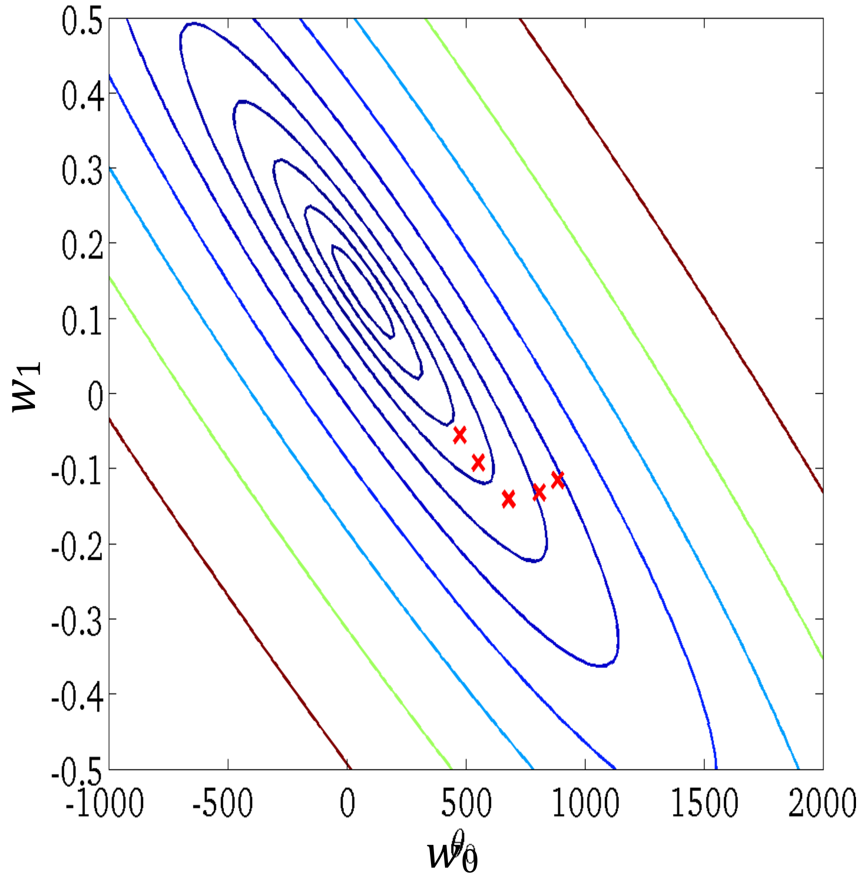
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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

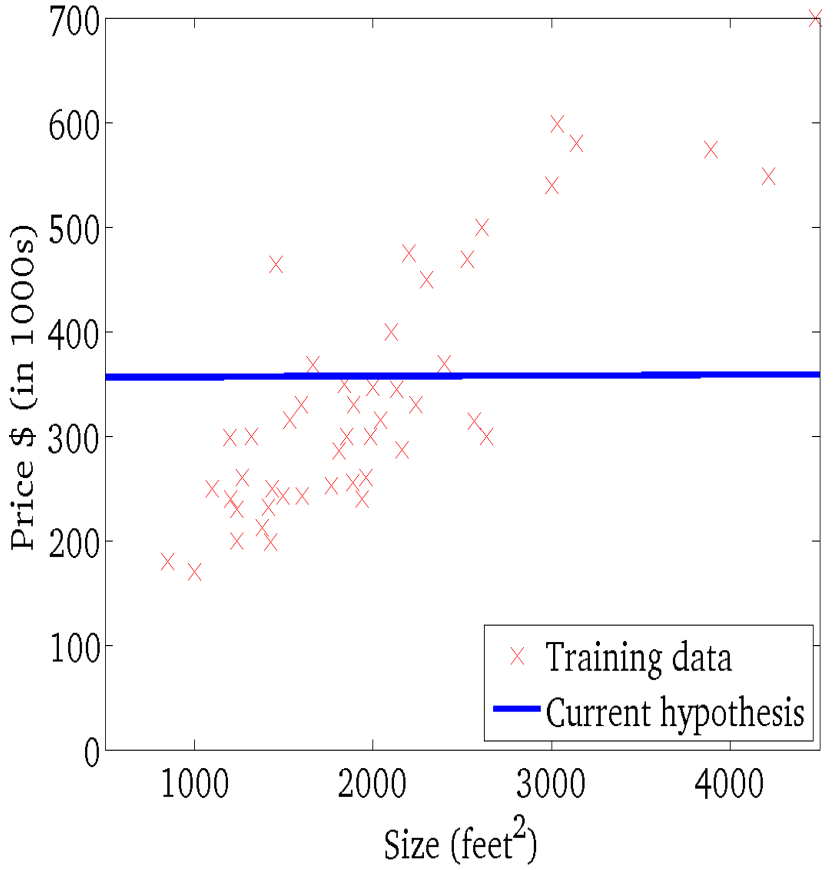


$J(w_0, w_1)$

(function of the parameters w_0,w_1)

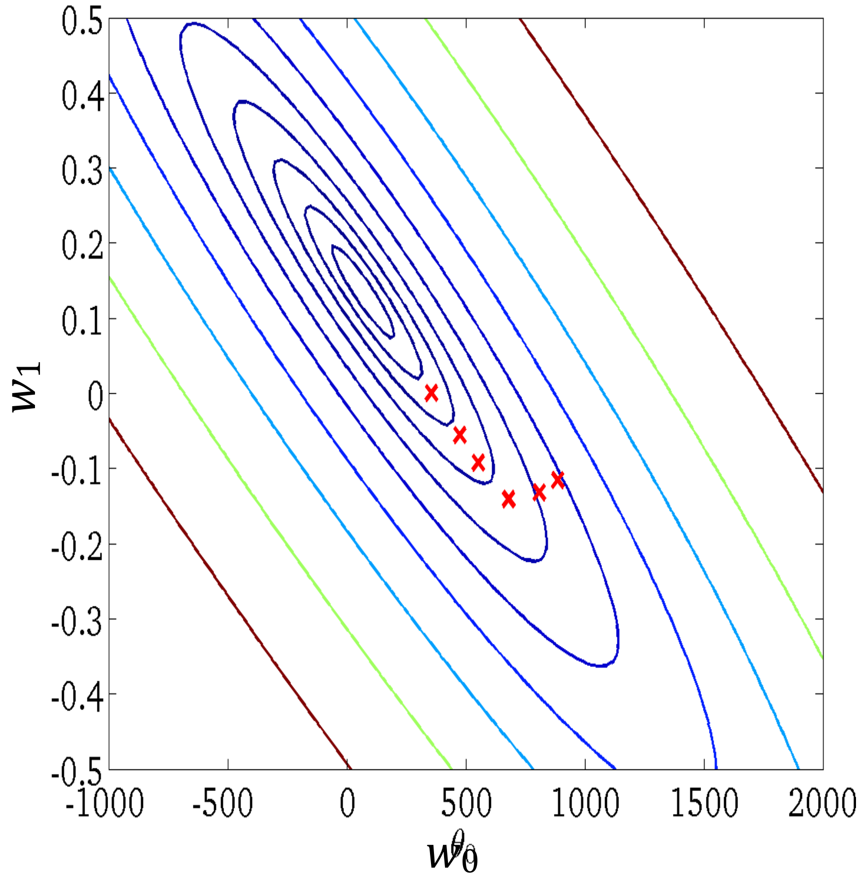


$$f(x; w_0, w_1) = w_0 + w_1 x$$

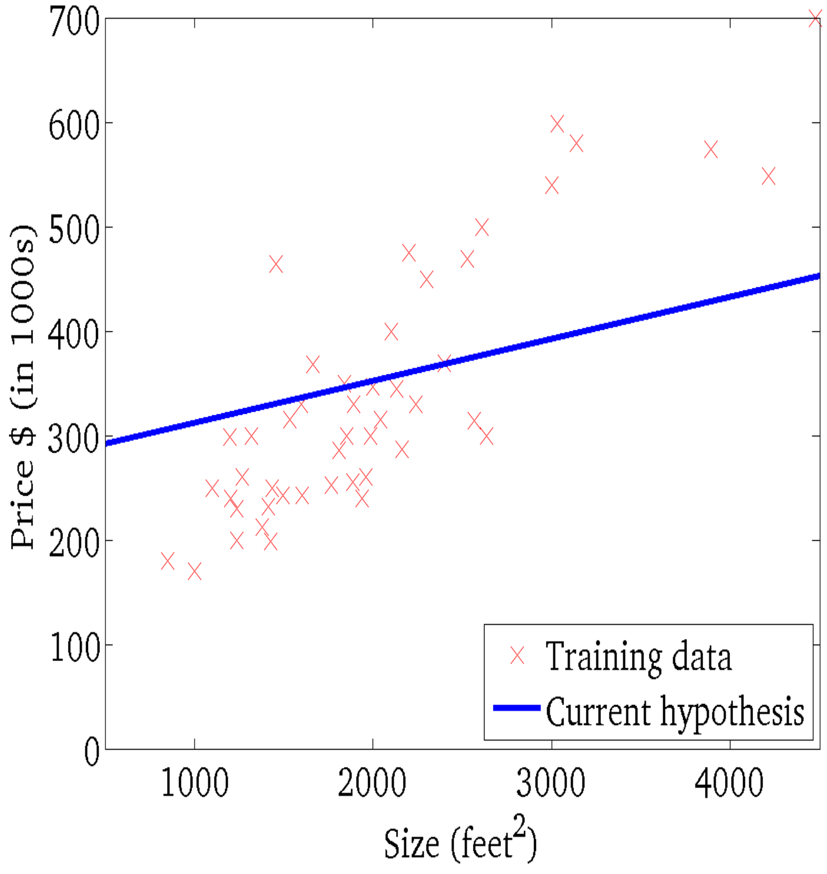


$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)

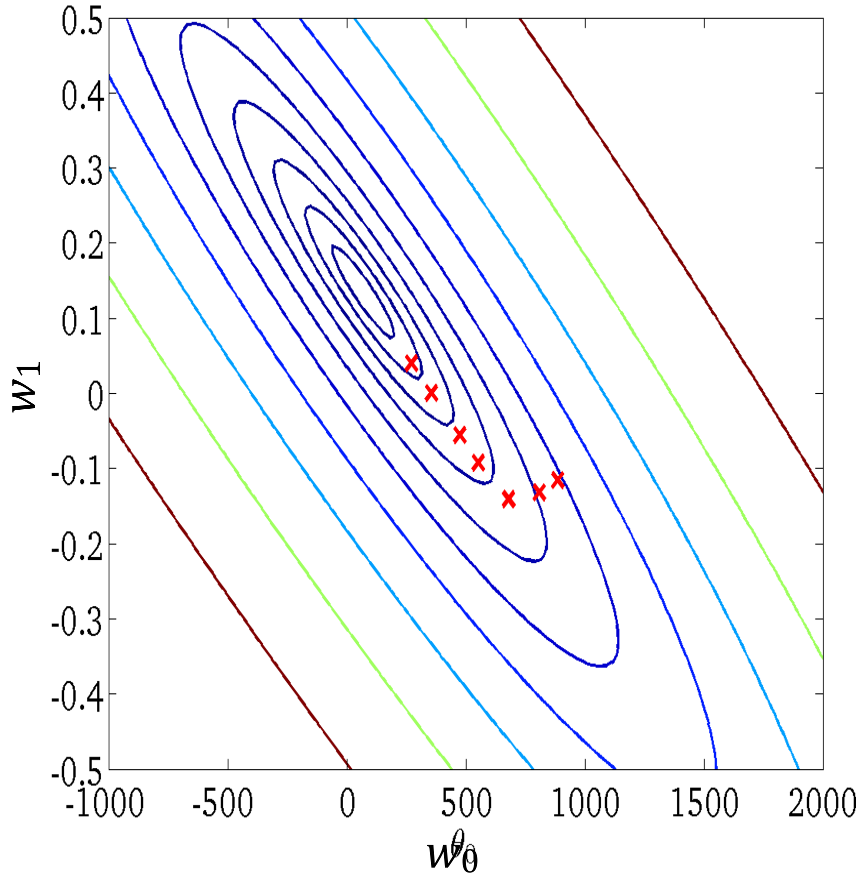


$f(x; w_0, w_1) = w_0 + w_1 x$



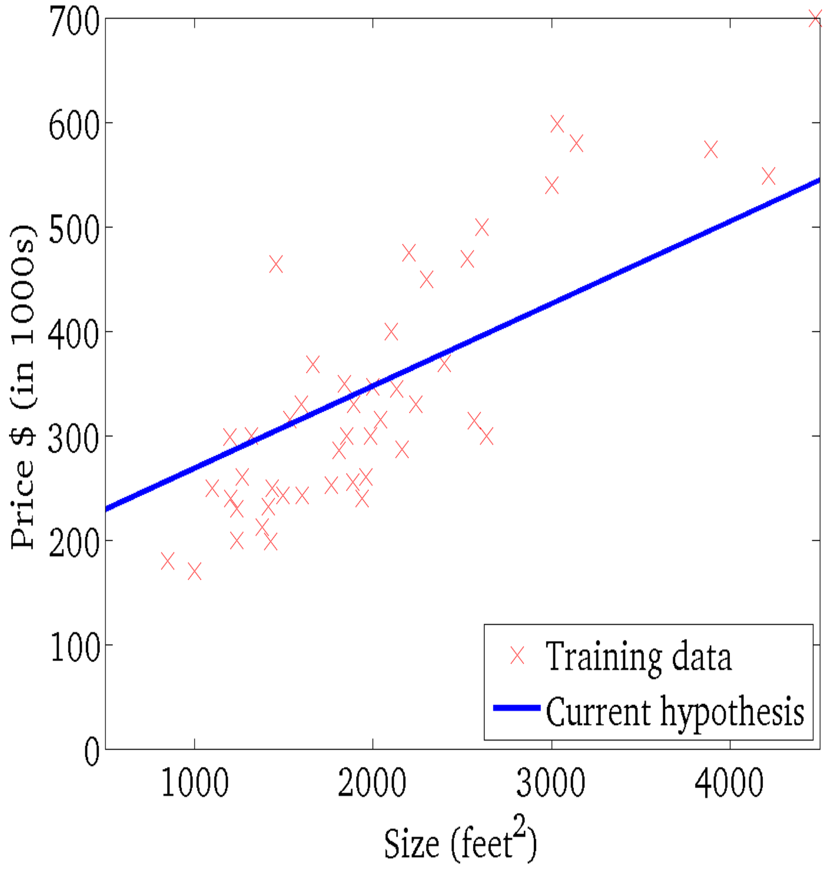
$J(w_0, w_1)$

(function of the parameters w_0,w_1)



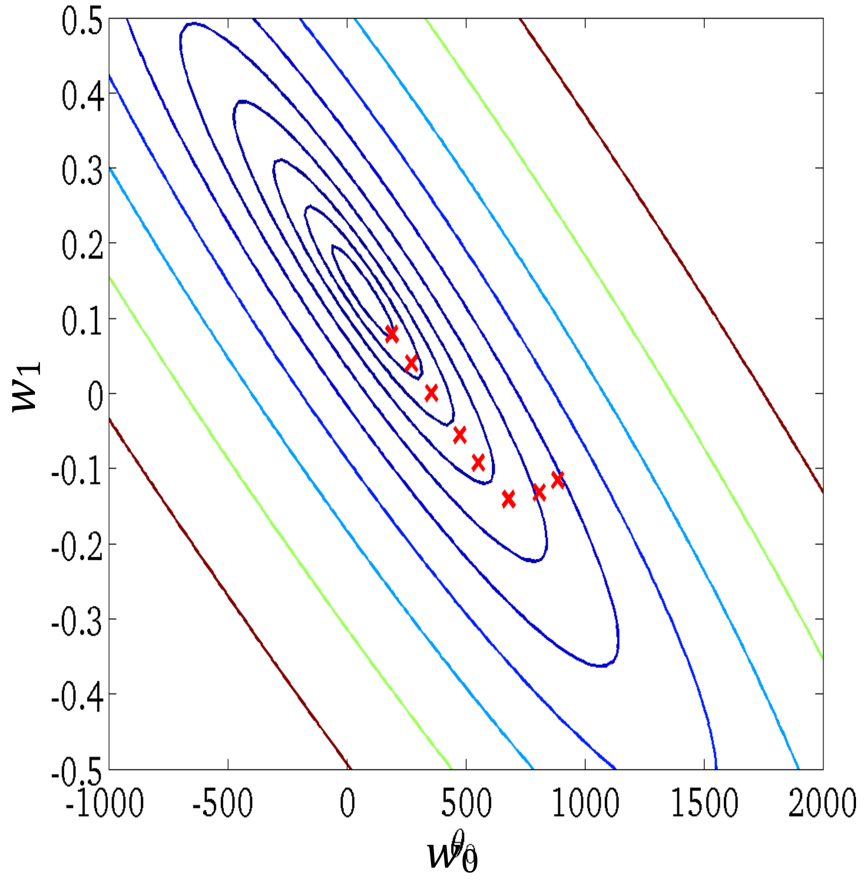
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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

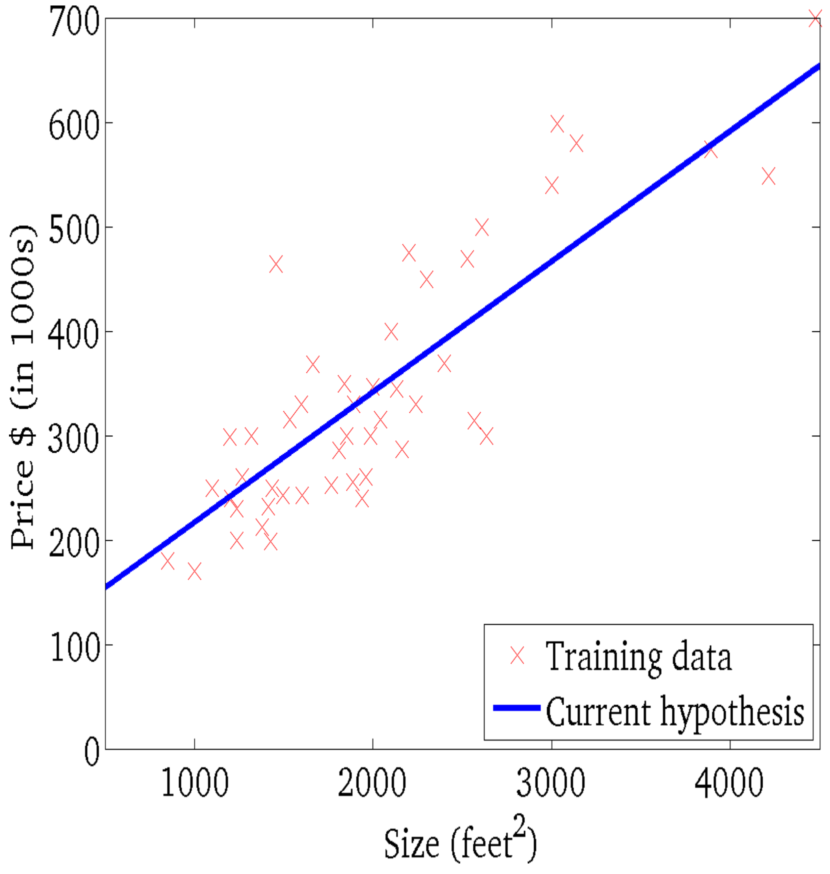


$$J(w_0, w_1)$$

(function of the parameters w_0,w_1)

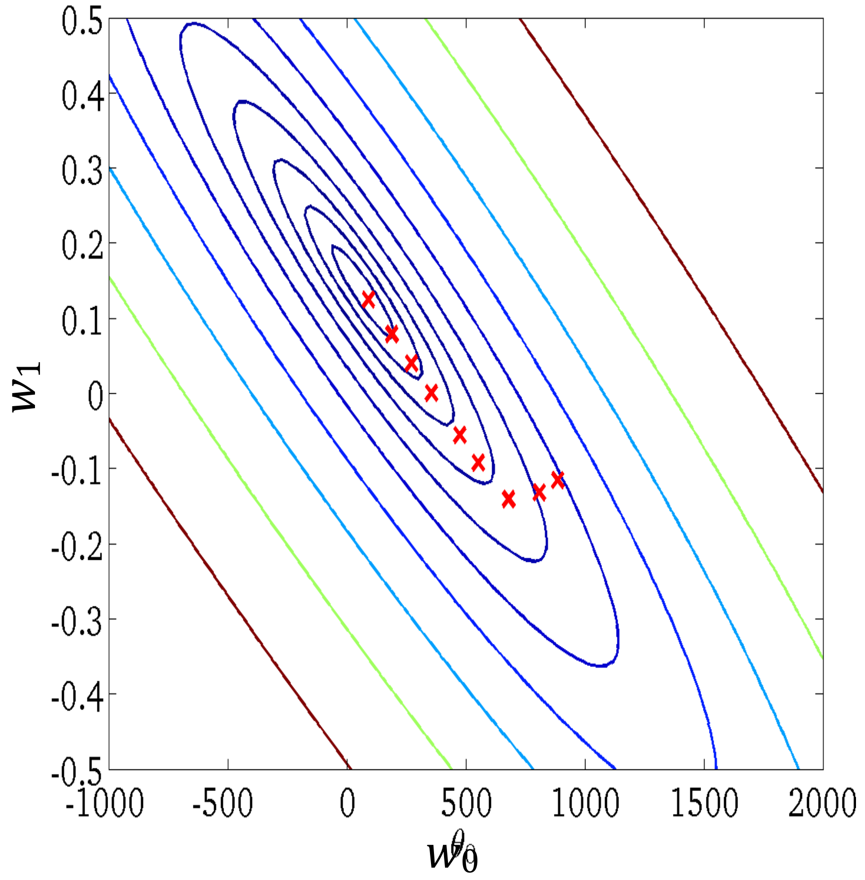


$f(x; w_0, w_1) = w_0 + w_1 x$



$J(w_0, w_1)$

(function of the parameters w_0,w_1)



Stochastic gradient descent

- Example: Linear regression with SSE cost function

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

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

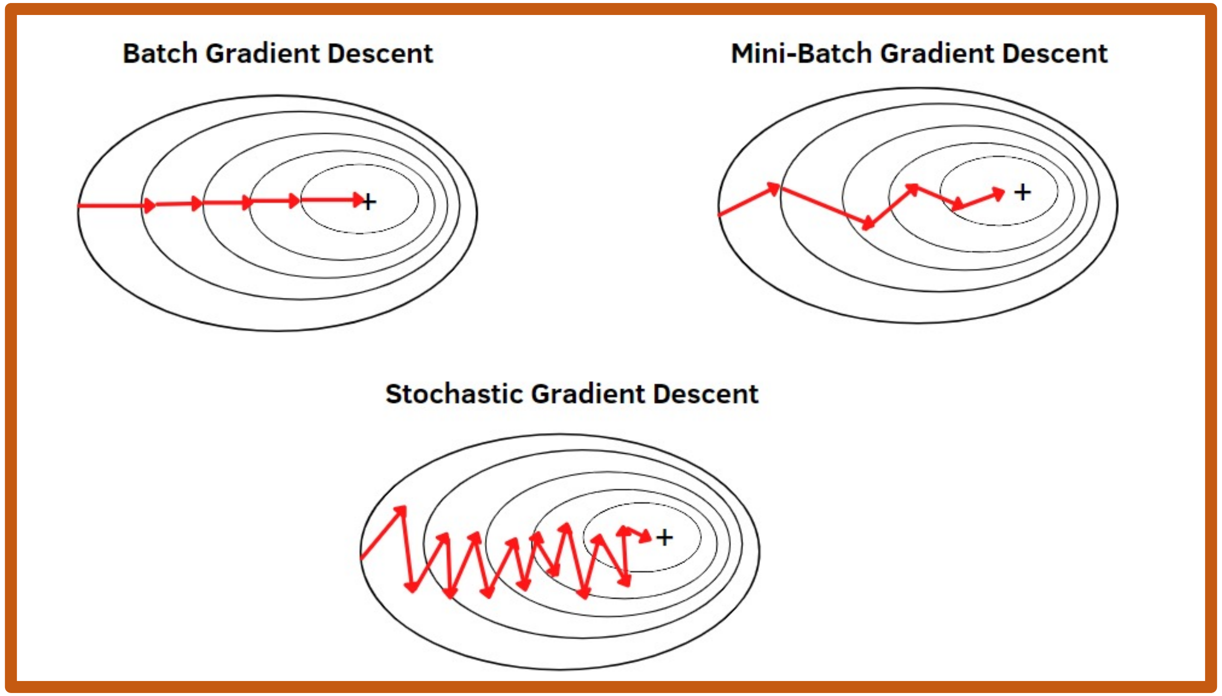
$$\boxed{\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta\left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right) \boldsymbol{x}^{(i)}}$$

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

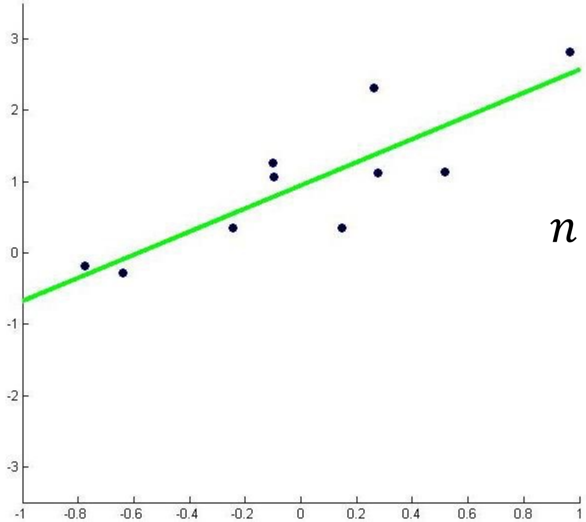
- Assumption: training and test examples are drawn independently at random from the same but unknown distribution.
  - Each training/test example $(\boldsymbol{x}, y)$ is a sample from joint probability distribution $P(\boldsymbol{x}, y)$, i.e., $(\boldsymbol{x}, y) \sim P$

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

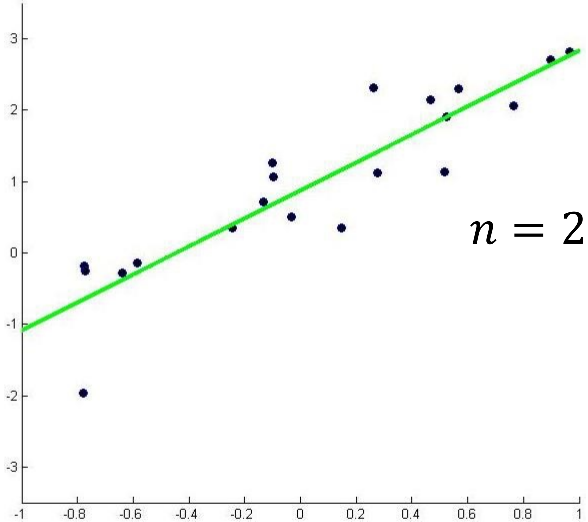
Expected (test) loss = $E_{\boldsymbol{x},y}\left\{Loss(y, f(\boldsymbol{x}; \boldsymbol{\theta}))\right\}$

- 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.

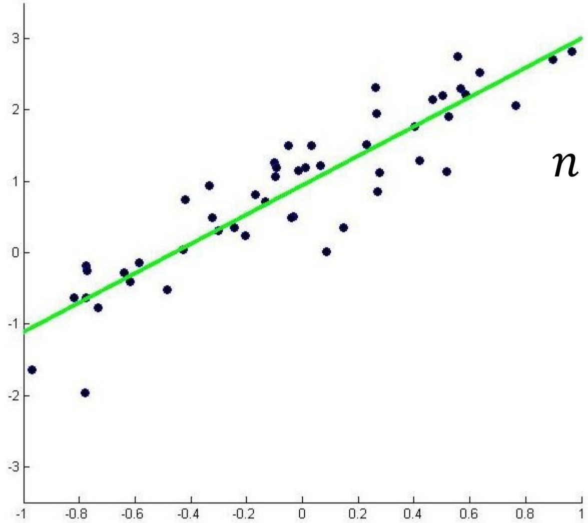
Linear regression: number of training data



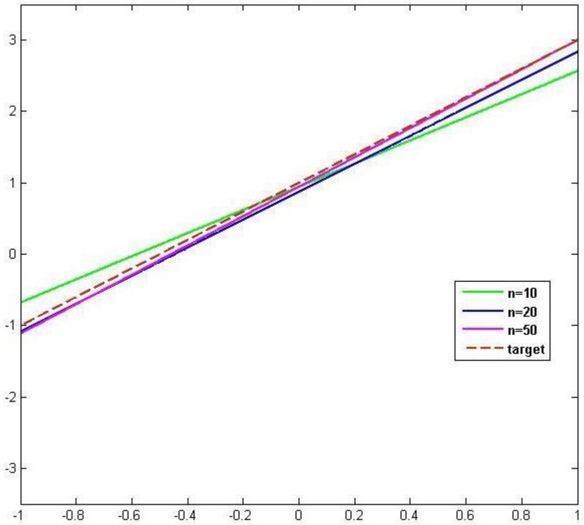
$n = 10$



$n = 20$

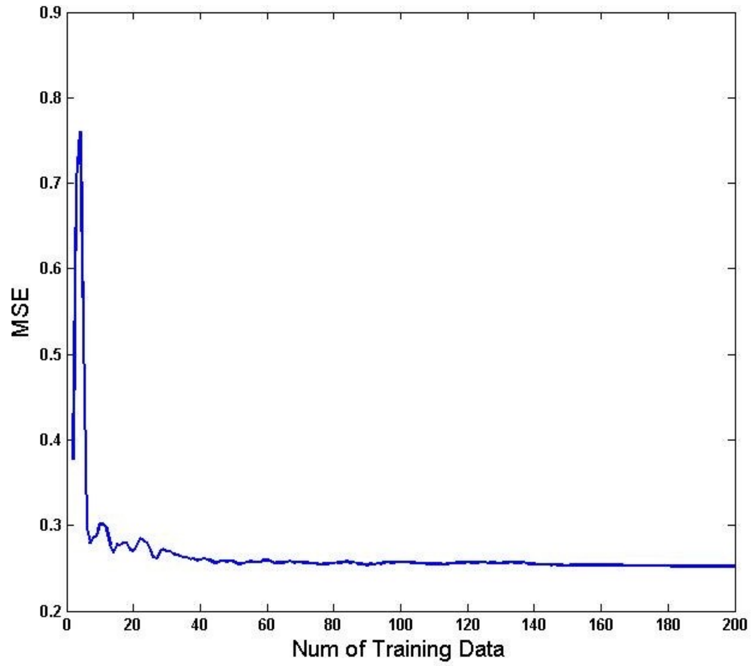


$n = 50$



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

- Structural error: the error introduced by the limited function class (infinite training data):

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

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

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

Linear regression: types of errors

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

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

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

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

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

Recall: Linear regression (squared loss)

- Linear regression functions

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

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

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

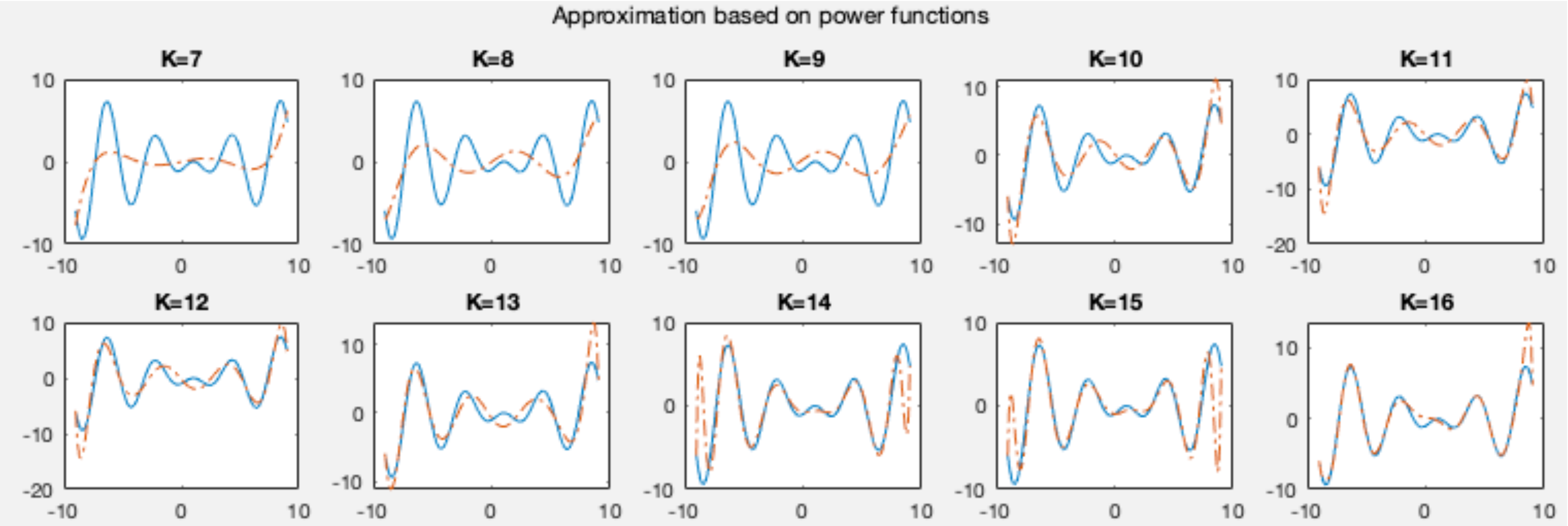
- Minimizing the squared loss for linear regression

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

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

Beyond linear regression

- How to extend the linear regression to non-linear functions?
  - Transform the data using basis functions
  - Learn a linear regression on the new feature vectors (obtained
  - by basis functions)



Beyond linear regression

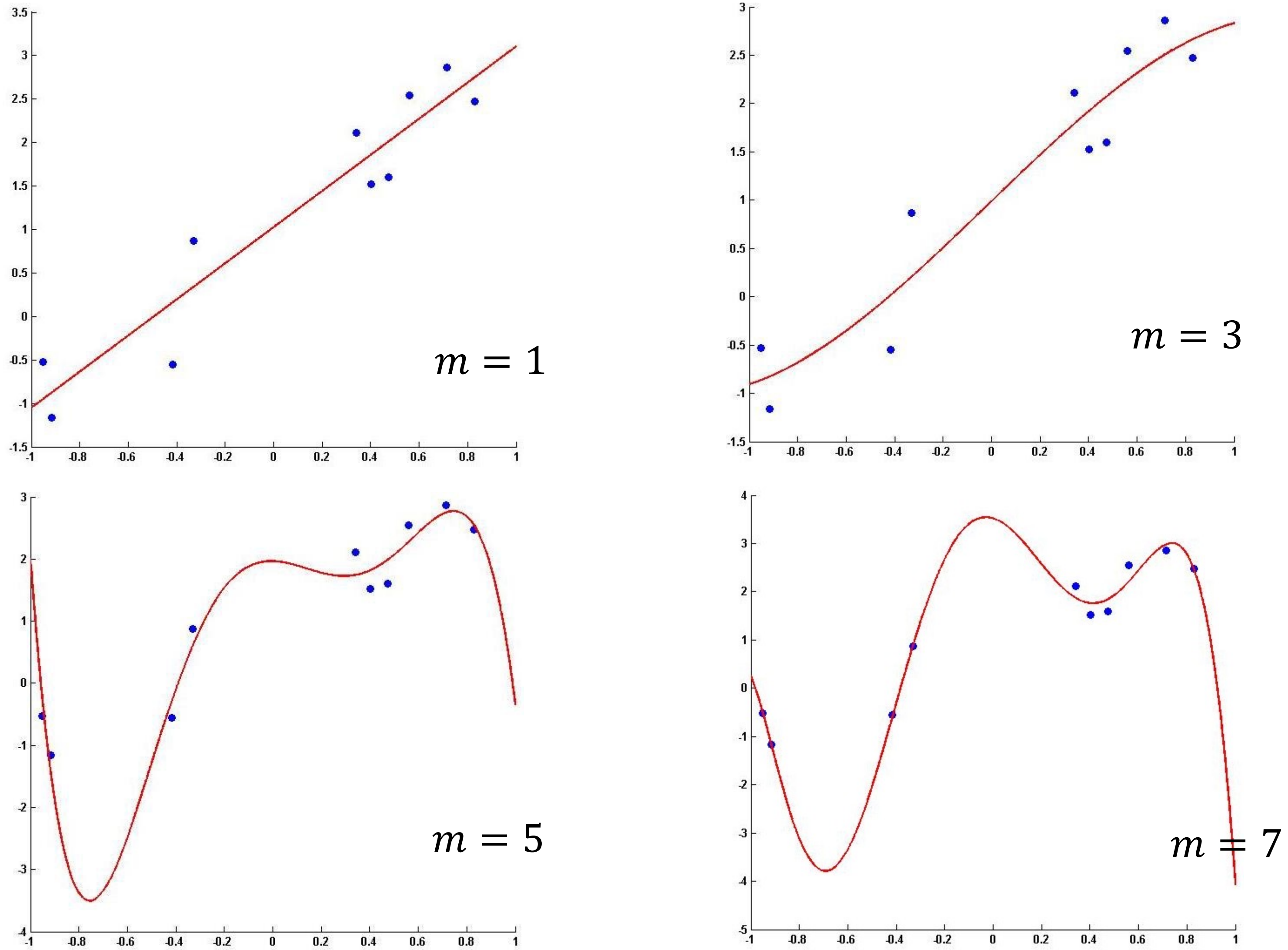
- m^{th} order polynomial regression (univariate $f : \mathbb{R} \longrightarrow \mathbb{R}$)

$$f(x; \boldsymbol{w}) = w_0 + w_1 x + \ldots + w_{m-1} x^{m-1} + w_m x^m$$

- Solution: $\boldsymbol{w} = \left(\boldsymbol{X}'^T \boldsymbol{X}'\right)^{-\mathbf{1}} \boldsymbol{X}'^T \boldsymbol{y}$

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad \boldsymbol{X}' = \begin{bmatrix} 1 & x^{(1)^1} & x^{(1)^2} & \cdots & x^{(1)^m} \\ 1 & x^{(2)^1} & x^{(2)^2} & \cdots & x^{(2)^m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x^{(n)^1} & x^{(n)^2} & \cdots & x^{(n)^1} \end{bmatrix} \boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}_0 \\ \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_m \end{bmatrix}$$

Polynomial regression: example



Generalized linear

- Linear combination of fixed non-linear function of the input vector

$$f(\boldsymbol{x};\boldsymbol{w}) \ = w_0 + w_1\phi_1(\boldsymbol{x}) + \ldots w_m\phi_m(\boldsymbol{x})$$

$\{\phi_1(\boldsymbol{x}),\ldots,\phi_m(\boldsymbol{x})\}$: set of basis functions (or features)

$$\phi_i(\boldsymbol{x}): \mathbb{R}^d \rightarrow \mathbb{R}$$

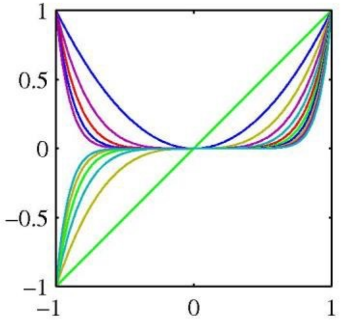
Basis functions: examples

- Linear

If $m = d$, $\phi_i(\mathbf{x}) = x_i$, $i = 1, \ldots, d$, then

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

- Polynomial (univariate)



If $\phi_i(x) = x^i$, $i = 1, \ldots, m$, then

$$f(x; \mathbf{w}) = w_0 + w_1 x + \ldots + w_{m-1} x^{m-1} + w_m x^m$$

Generalized linear: optimization

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

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

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \boldsymbol{\Phi} = \begin{bmatrix} 1 & \phi_1(\boldsymbol{x}^{(1)}) & \cdots & \phi_m(\boldsymbol{x}^{(1)}) \\ 1 & \phi_1(\boldsymbol{x}^{(2)}) & \cdots & \phi_m(\boldsymbol{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(\boldsymbol{x}^{(n)}) & \cdots & \phi_m(\boldsymbol{x}^{(n)}) \end{bmatrix} \; \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

$$\boldsymbol{w} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} \right)^{-\mathbf{1}} \boldsymbol{\Phi}^T \boldsymbol{y}$$

Resource

1 C. M. Bishop, *Pattern Recognition and Machine Learning*.

2 Y. S. Abu-Mostafa, “Machine learning.” California Institute of Technology, 2012.

3 Machine Learning, Dr. Soleymani, Sharif University