Universitat
de les Illes Balears

Machine Learning

Lesson 2: Supervised Learning

Linear Models (LMS, Logistic Regression, Perceptron)

Remember: a simple example…

Size (feet2)	*Number of bedrooms*	*Number of floors*	*Age of home (years)*	*Price ($1000)*
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

… for dataset notation

Dataset = $\{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, \ldots m\}$

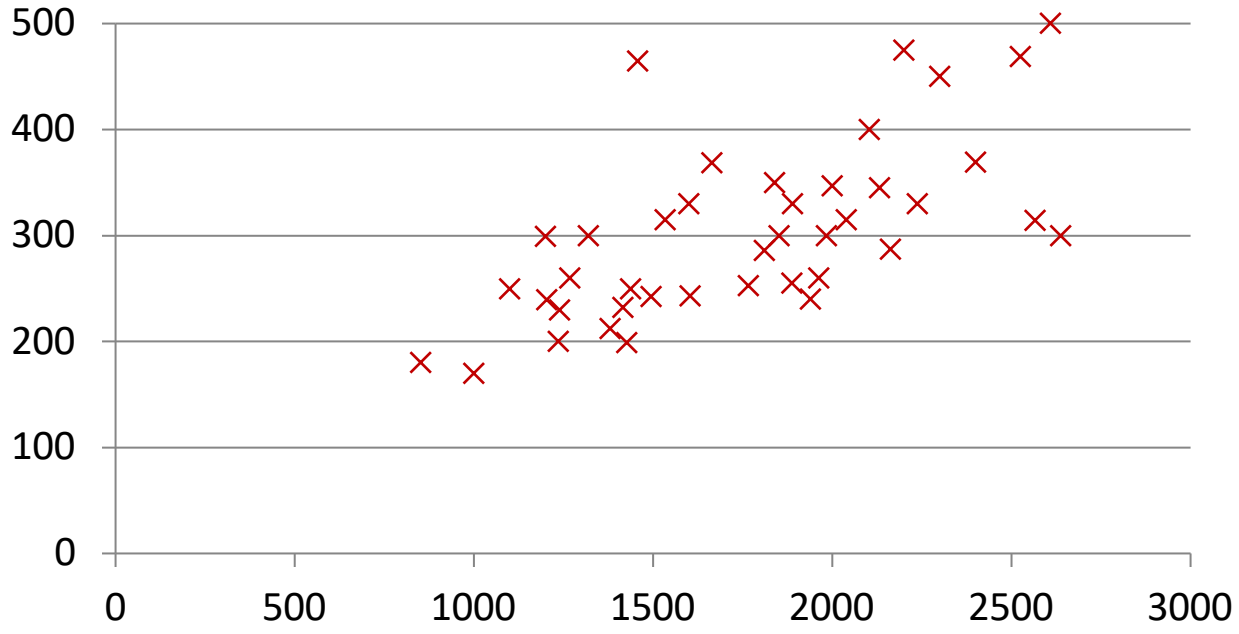
$(\mathbf{x}^{(i)}, y^{(i)})$ = *Training example*

$\mathbf{x}^{(i)}$= *"input" variable (features),* $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right)$

$y^{(i)}$ = *"output" variable (target),* $y \in \mathcal{Y}$

What price…?

Size (feet²)	Price ($) in 1000's
2104	460
1416	232
1534	315
852	178
…	…



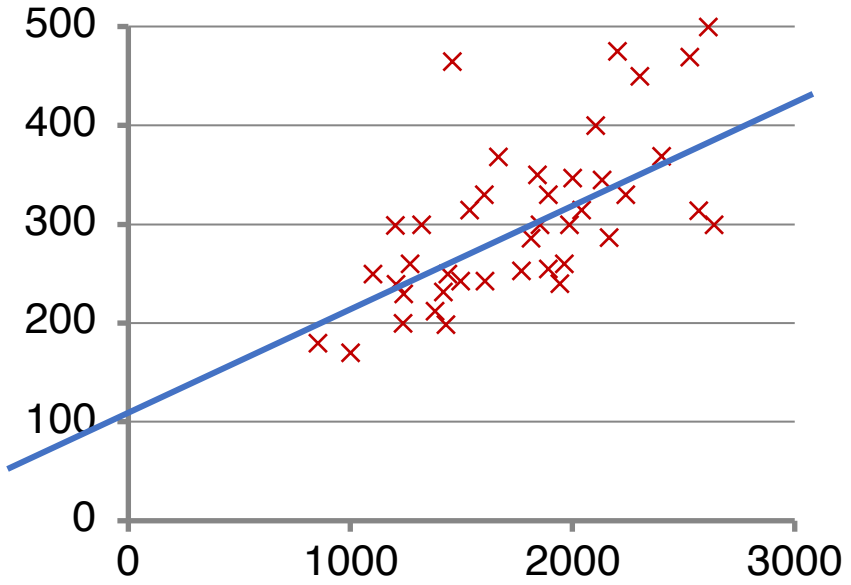
ML Model: regression

Size (feet²)	Price ($) in 1000's
2104	460
1416	232
1534	315
852	178
…	…

Linear regression

Hypothesis: the model is linear

$$h_\theta(x) = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$



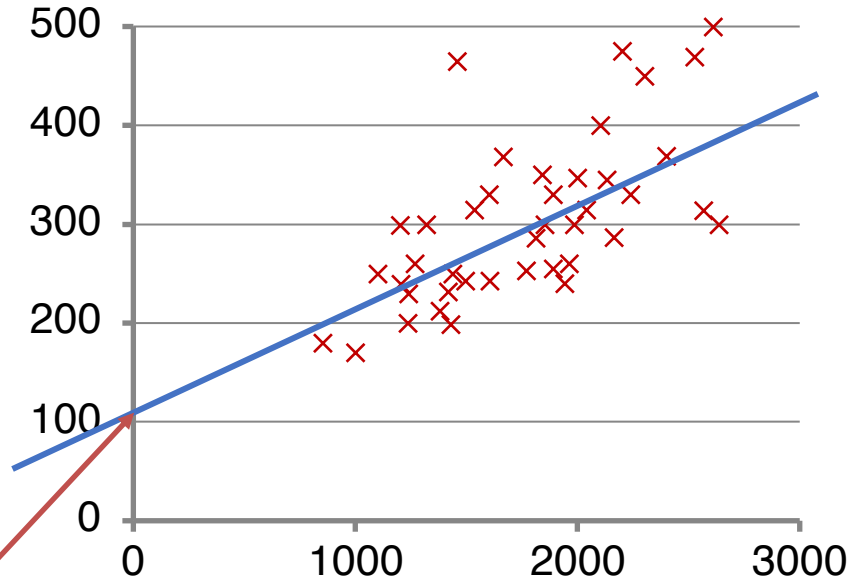
Case n=1:

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

Linear regression

Hypothesis: the model is linear

$$h_\theta(x) = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$



is called the **bias** because it represents the intercept or prior (prejudice)

Linear regression

Hypothesis: the model is linear

$$h_\theta(x) = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

n = number of input dimensions

Generalization

(in matrix notation)

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

x_0=1 (Intercept term)

Loss function

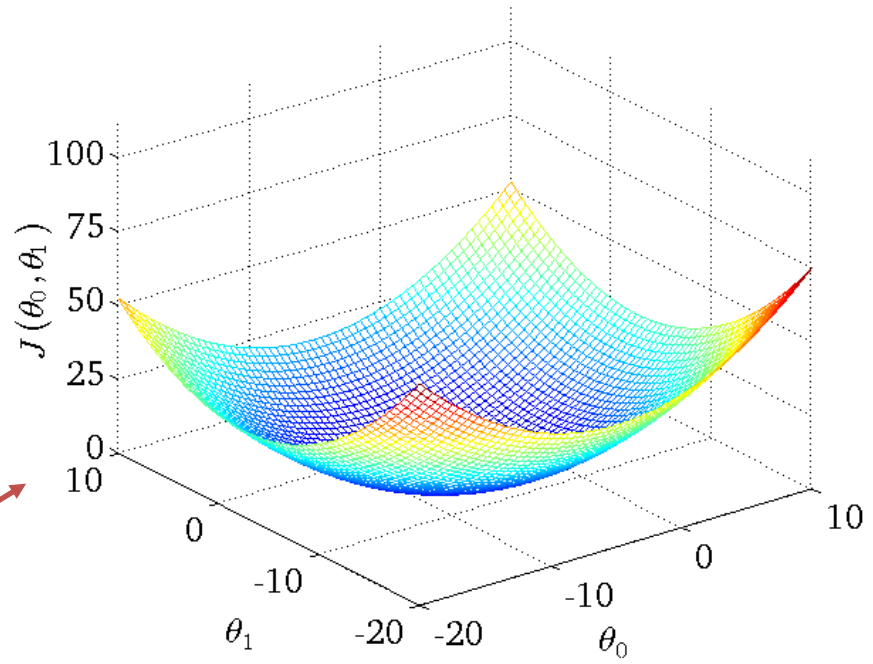
Given a training set, how do we learn the parameters?

Basic idea: to make h_θ close to y

We've to define a function that measures, for each parameters' values, how close the $h_\theta(x^{(i)})$'s are to the corresponding $y^{(i)}$'s

Quadratic loss function

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



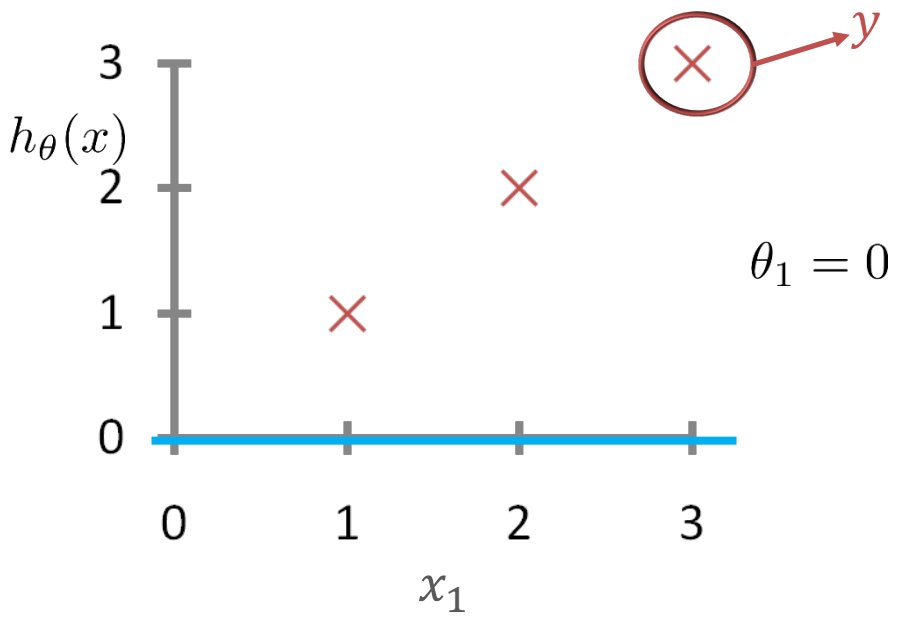
Case n=1:

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

Quadratic loss function: example

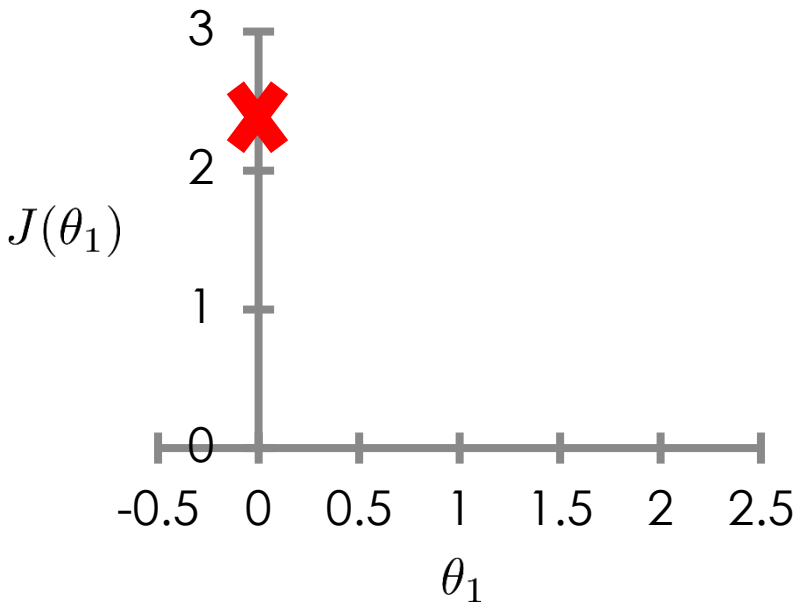
$h_\theta(x) = \theta_1 x_1$

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



$J(\theta_1)$

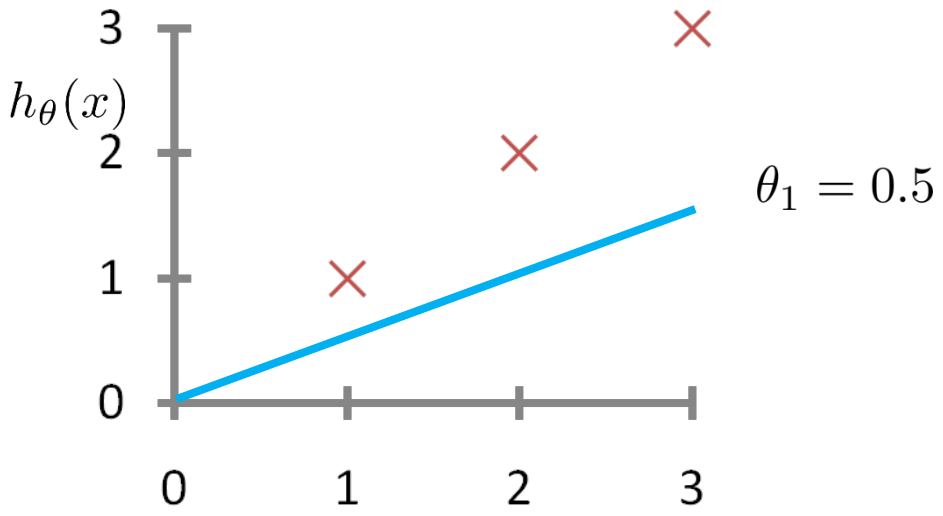
(function of the parameter θ_1)



Quadratic loss function: example

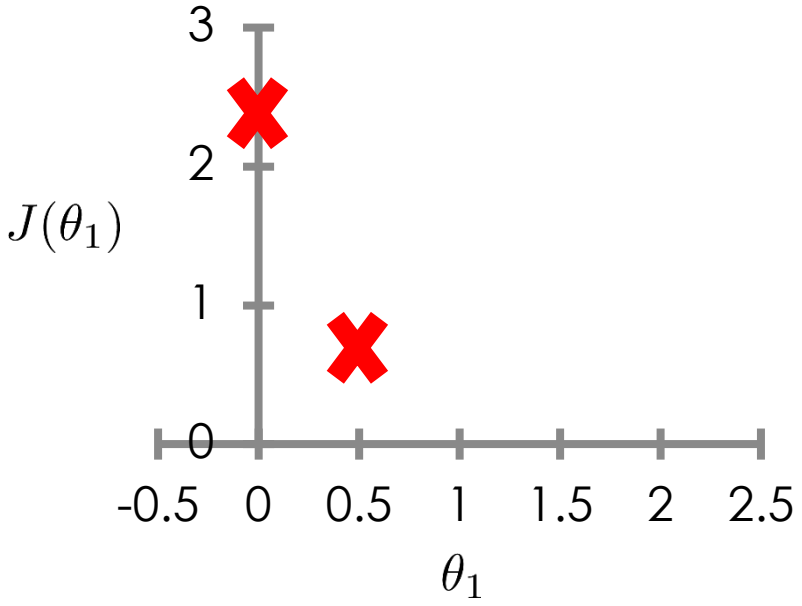
$h_\theta(x) = \theta_1 x_1$

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



$J(\theta_1)$

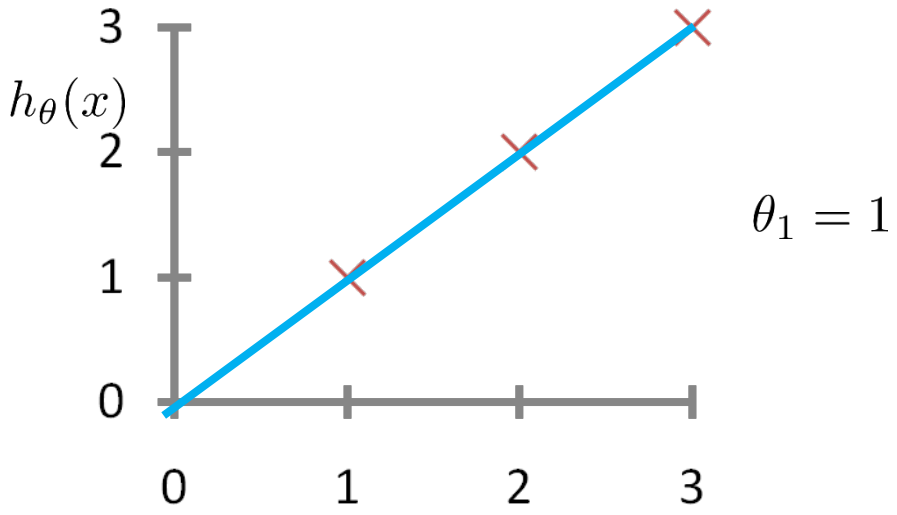
(function of the parameter θ_1)



Quadratic loss function: example

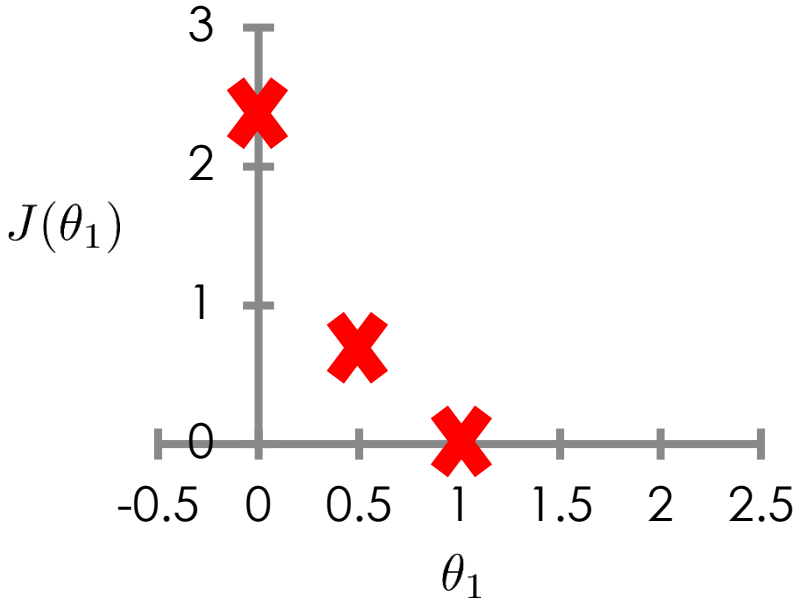
$h_\theta(x) = \theta_1 x_1$

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



$J(\theta_1)$

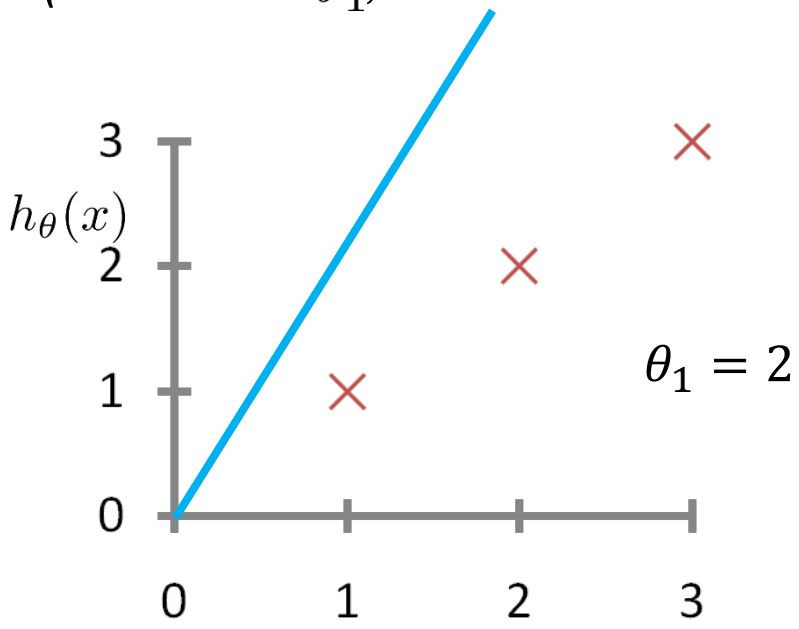
(function of the parameter θ_1)



Quadratic loss function: example

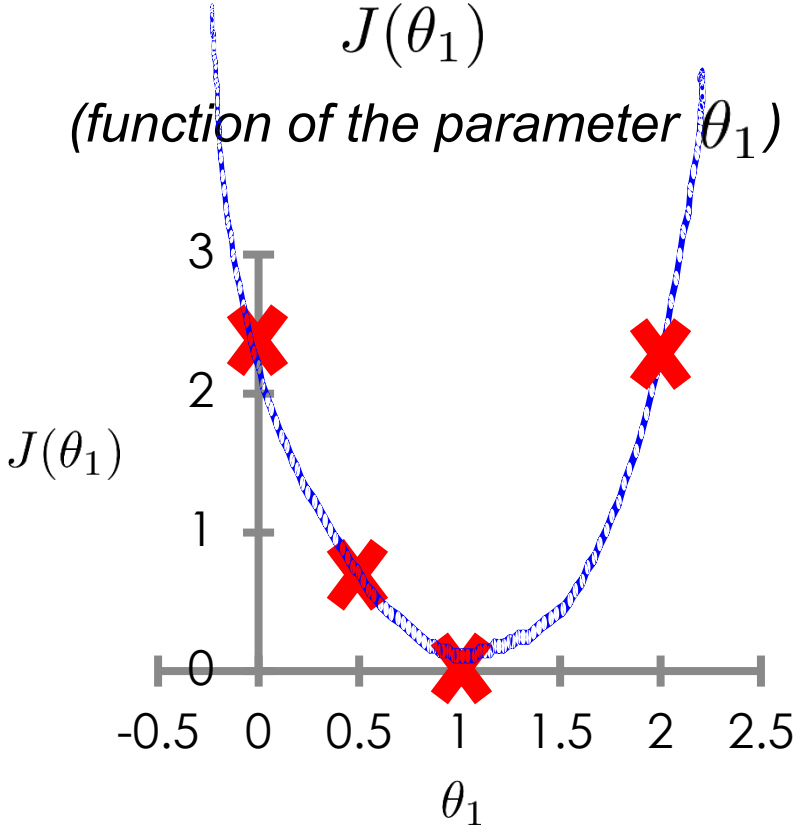
$h_\theta(x) = \theta_1 x_1$

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



$J(\theta_1)$

(function of the parameter θ_1)



Analytic Linear regression

Dataset = $X, \vec{y}$, where X is a matrix $m \times (n+1)$

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

$$\vec{y} = X\theta$$

$$X\theta = \vec{y}$$

$$X^T X\theta = X^T\vec{y}$$

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

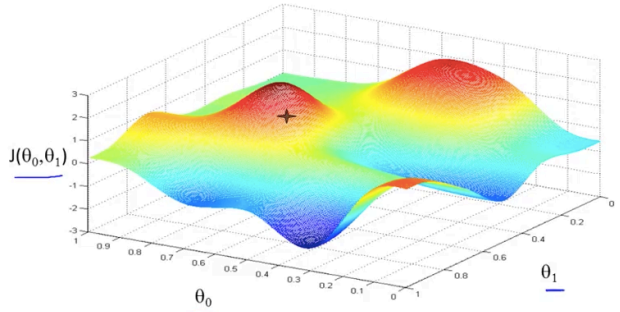
Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

full derivation

LMS learning algorithm

Loss function: Quadratic

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

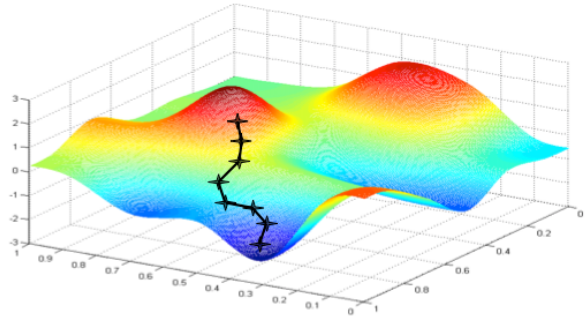


Goal: To minimize $J(\theta)$

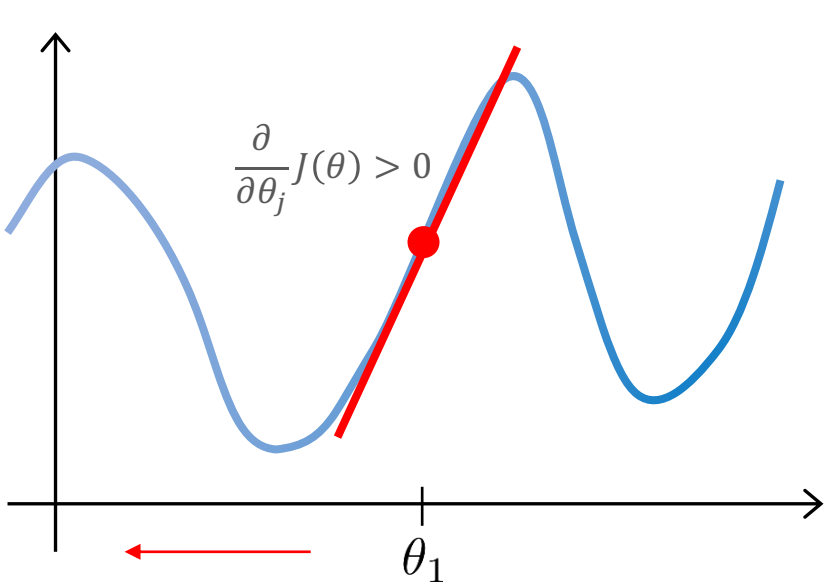
Gradient descent

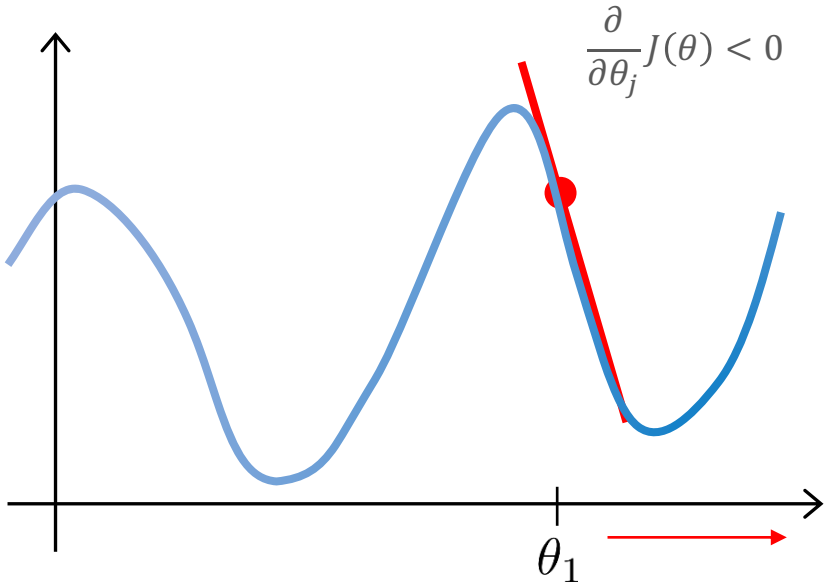
$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

Learning rate

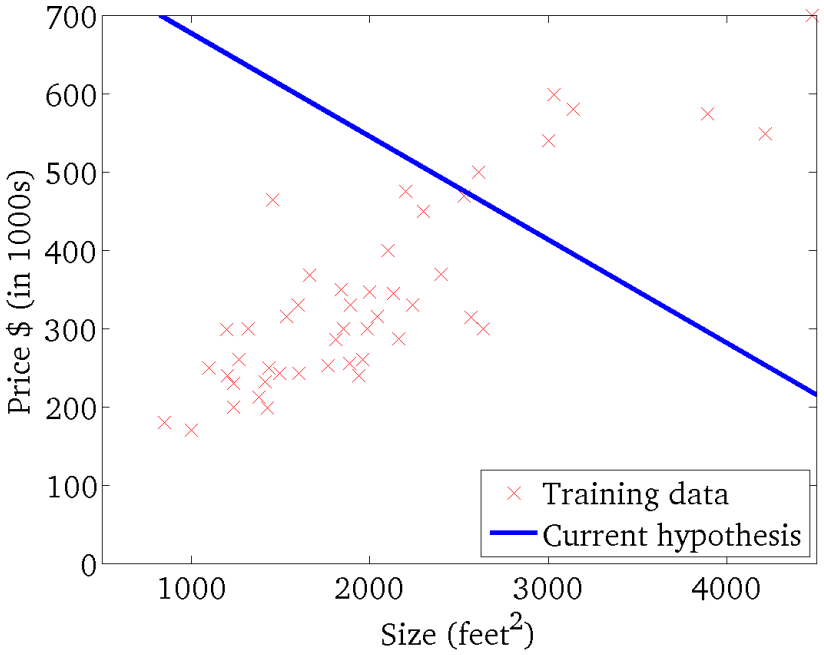


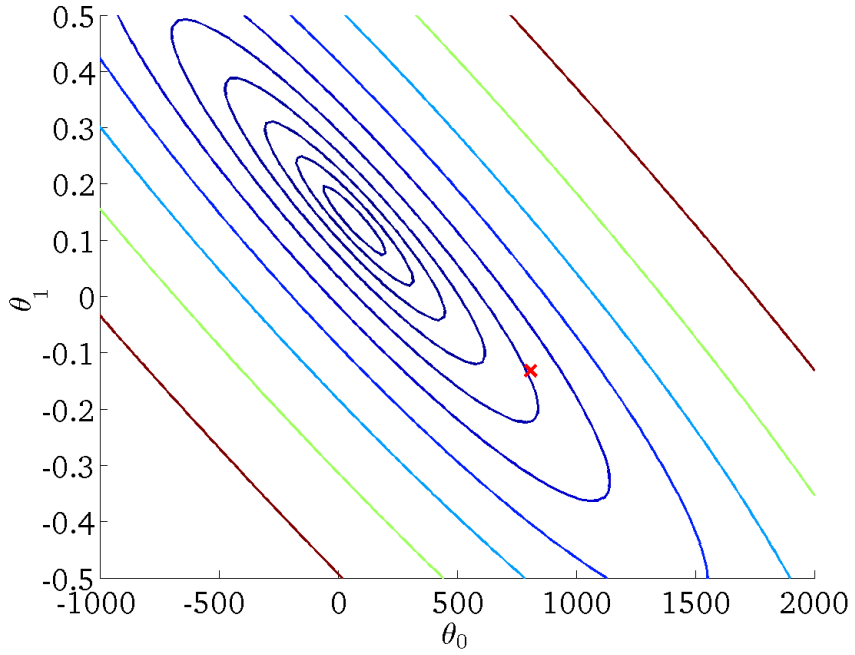
Gradient descent: basic idea



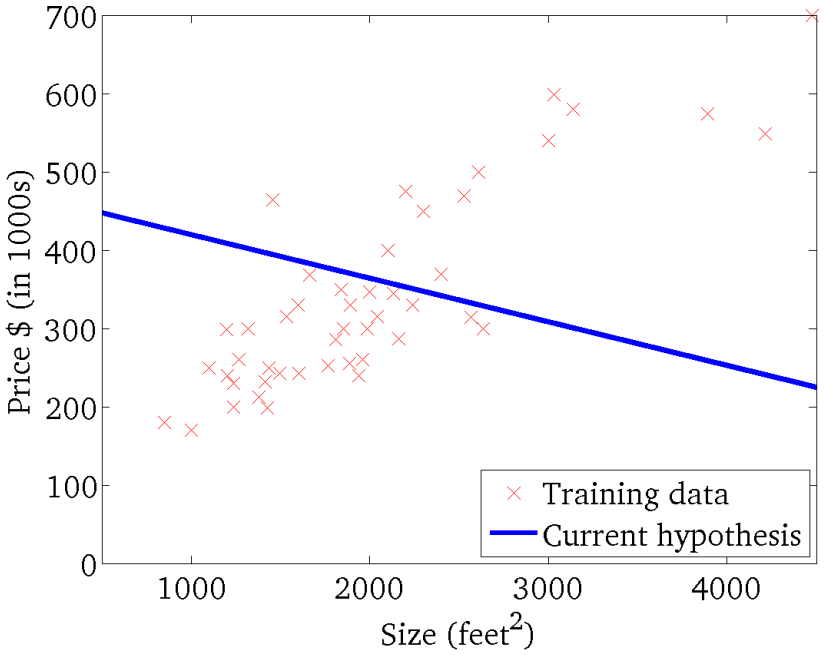


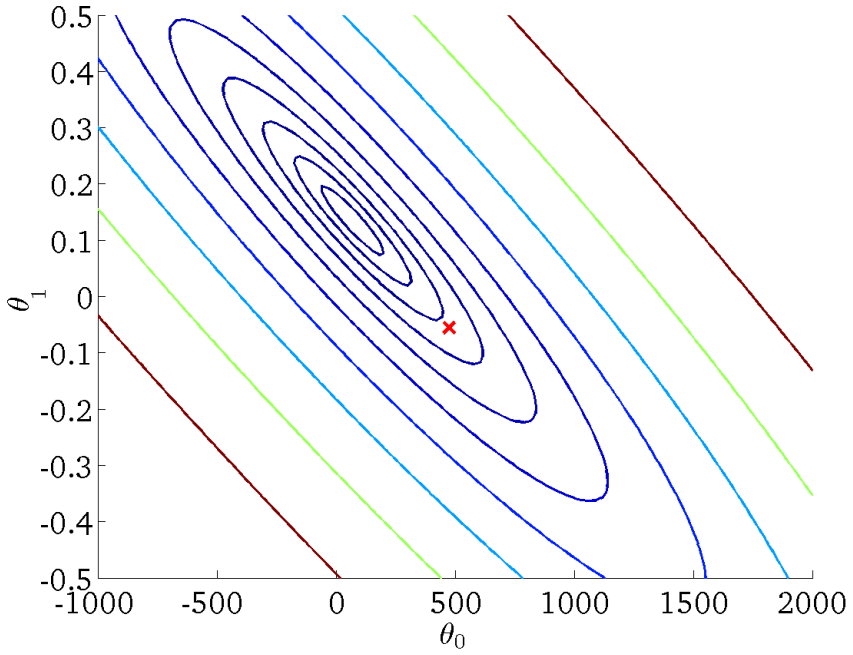
Gradient descent: Linear



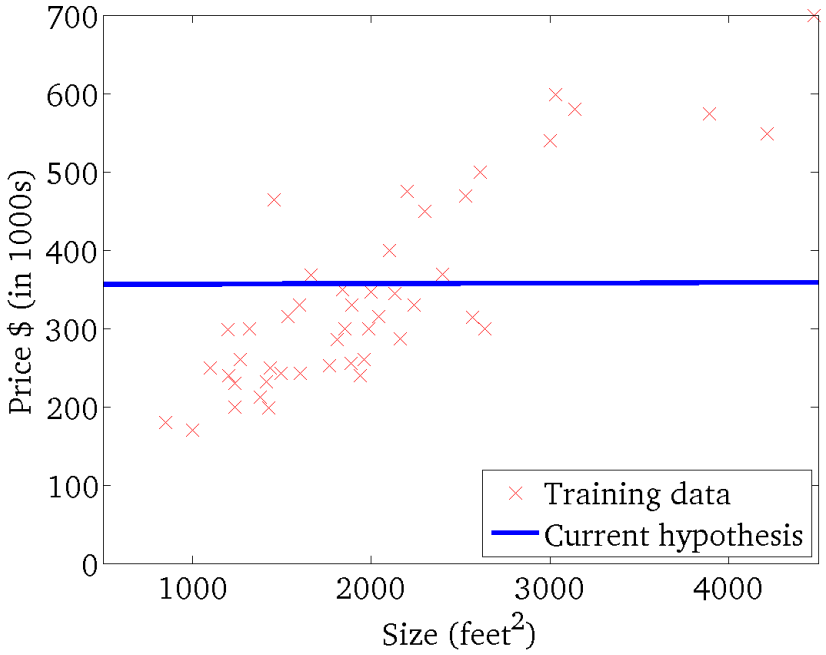


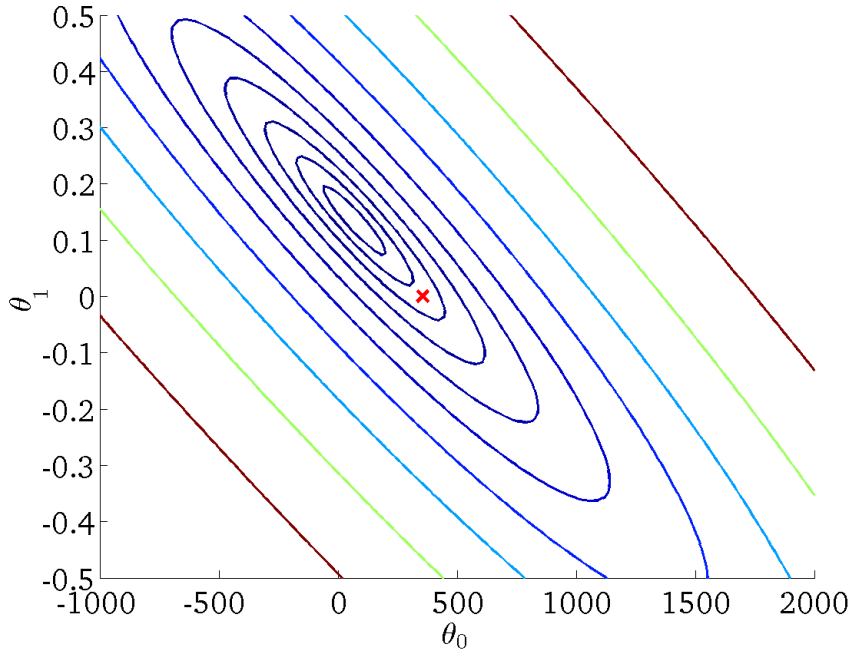
Gradient descent: Linear



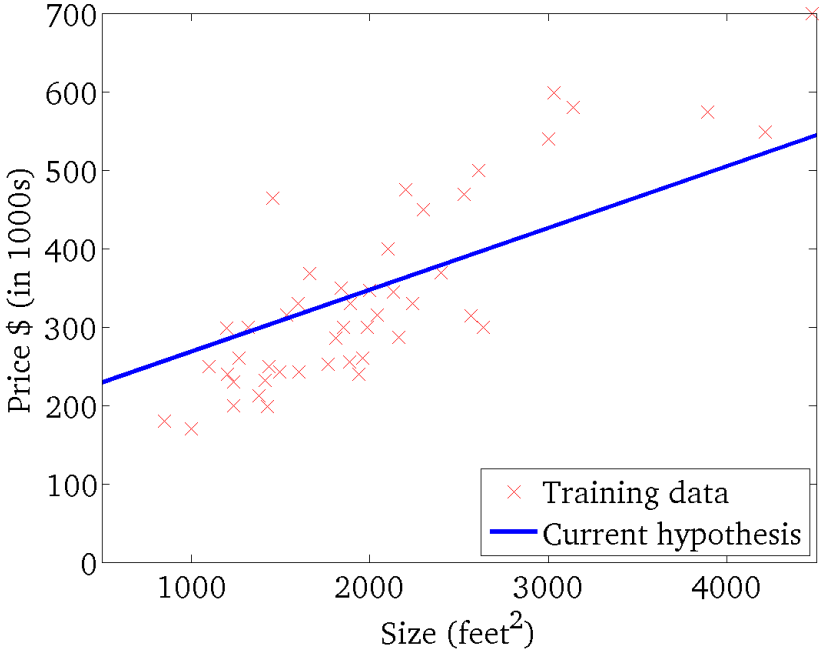


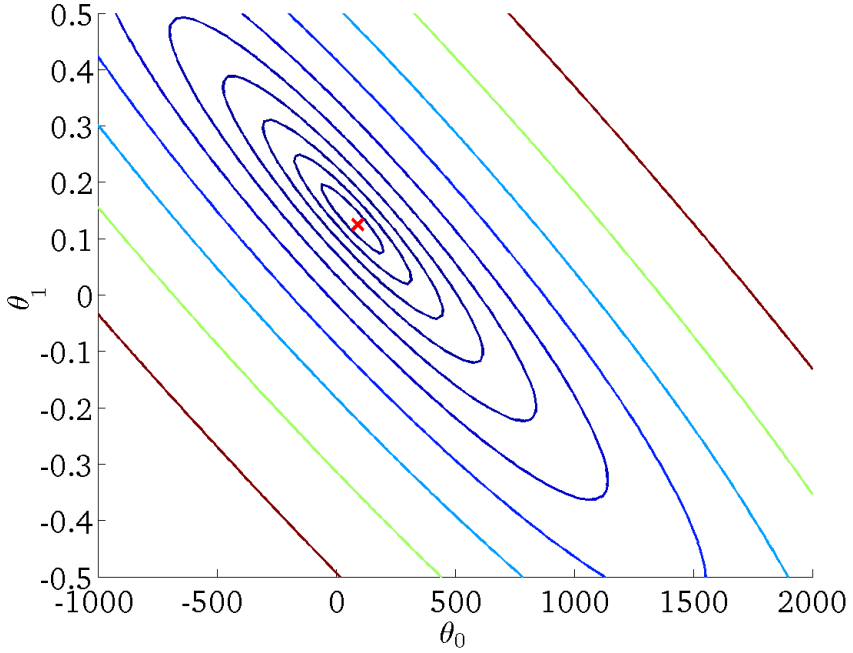
Gradient descent: Linear





Gradient descent: Linear





LMS algorithm: update rule

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta) \longrightarrow \theta_j := \theta_j + \alpha \cdot \left(y - h_\theta(x)\right) x_j$$

$m = 1$

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

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2$$

$$= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y)$$

$$= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_j x_j - y \right)$$

$$= (h_\theta(x) - y) x_j$$

This rule (also known as Widrow-Hoff learning rule) has several properties that seem natural and intuitive. For instance, the magnitude of the update is proportional to the error term $\left(y - h_\theta(x)\right)$; thus, for instance, if we are encountering a training example on which our prediction nearly matches the actual value, then, we find that there is little need to change the parameters; in contrast, a larger change to the parameters will be made if our prediction has a large error.

Learning rate

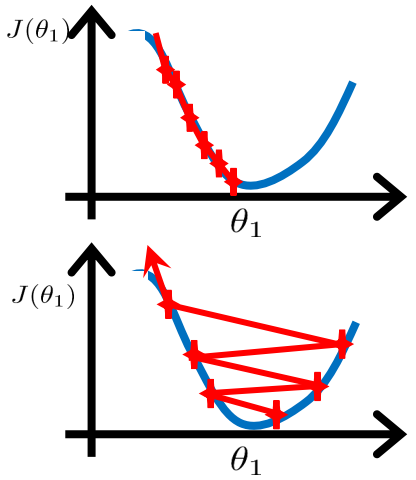
$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

Learning rate

In Practice:

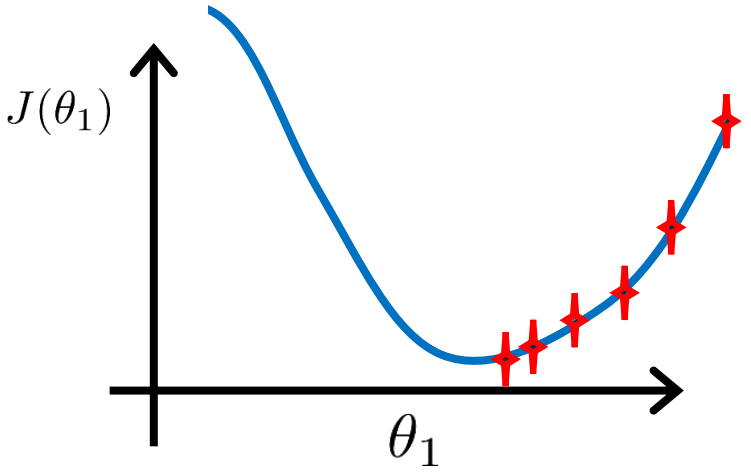
If a is too small, gradient descent can be slow.

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



Gradient descent can be susceptible to **local minima** in general, the optimization problem we have posed here for linear regression has only one global, and no other local, optima; thus gradient descent always converges (assuming the learning rate a is not too large) to the global minimum.

As we approach a local minimum, gradient descent will automatically take smaller steps, being [0,1] smooths the derivative by reducing the jump stride.

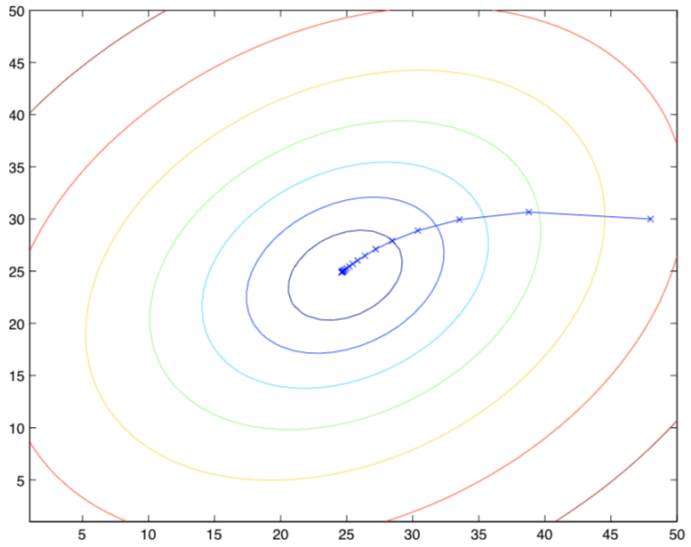


LMS algorithm: batch rule

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right) \right) x_j^{(i)}$$

} (for every j)



LMS algorithm: incremental rule

Loop{

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta\left(x^{(i)}\right) \right) x_j^{(i)}$$

}

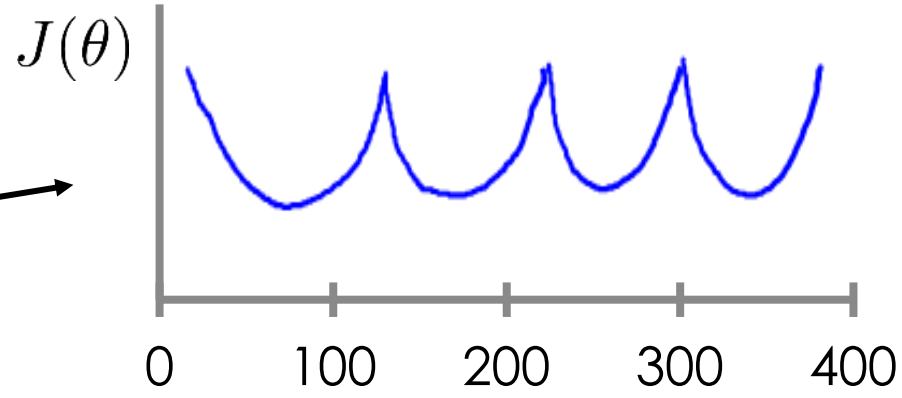
}

When the training set is large, **stochastic gradient descent** is often preferred over batch gradient descent

Whereas batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if m is large—stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at. Often, stochastic gradient descent gets θ "close" to the minimum much faster than batch gradient descent. (Note however that it may never "converge" to the minimum, and the parameters will keep oscillating around the global minimum; but in practice, most of the values near the minimum will be reasonably good approximations to the true minimum.

LMS in practice

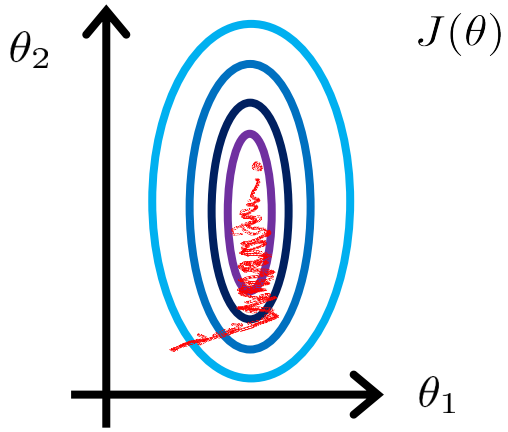
It is necessary to display the cost function

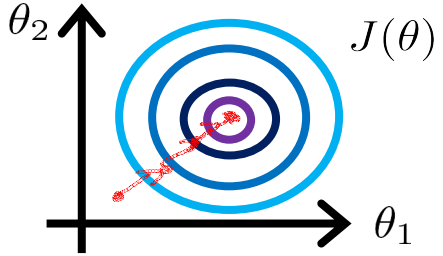


Simultaneous update

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

Data scaling / normalization





Summary of Linear models

Linear regression

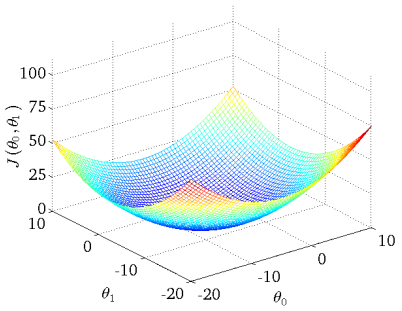
Hypothesis:

the model is linear

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

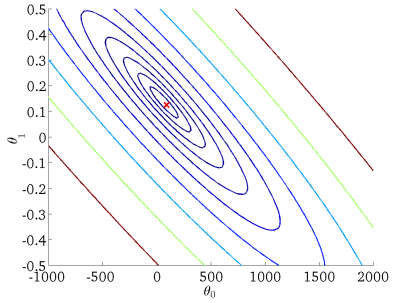
Gradient descent

Idea: *to make* h_θ *close to* y *by means minimizing a cost function*



LMS

Using the gradient to find the minimum (batch & incremental)

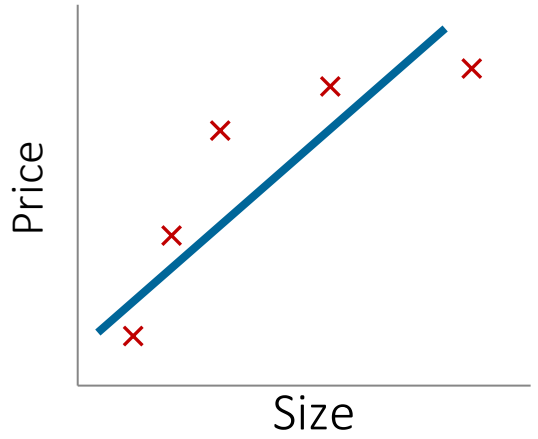


Bias-Variance Tradeoff

A learning algorithm is **biased** for a particular input (x) if, when trained on different data sets, it is systematically incorrect when predicting the correct output for x.

Therefore, the **bias** is an error from erroneous assumptions in the learning algorithm.

High bias can cause an algorithm to miss the relevant relations between features and target outputs (**underfitting**).



$$\theta_0 + \theta_1 x$$

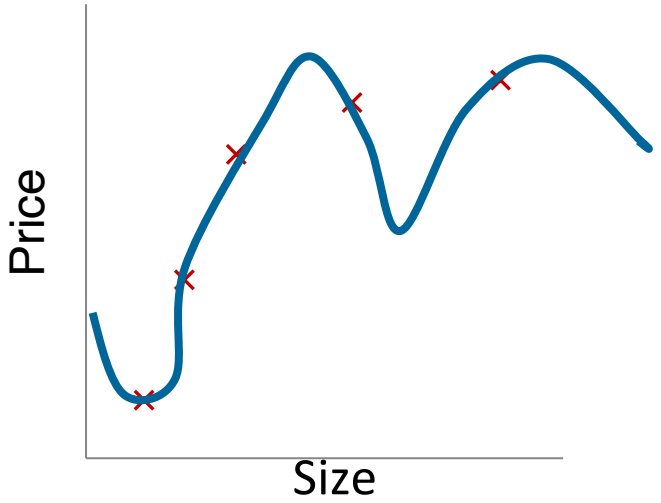
Bias
(Underfitting)

Bias-**Variance** Tradeoff

A learning algorithm has high **variance** for a particular input x if it predicts different output values when trained on different training sets.

The **variance** is an error from sensitivity to small fluctuations in the training set.

High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (**overfitting**).



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

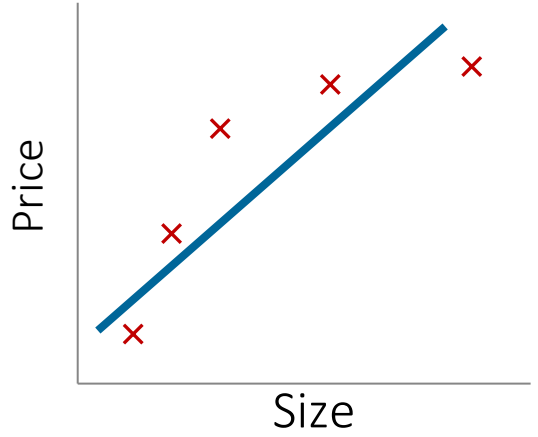
Variance
(Overfitting)

Bias-Variance Tradeoff

A learning algorithm is **biased** for a particular input (x) if, when trained on different data sets, it is systematically incorrect when predicting the correct output for x.

Therefore, the **bias** is an error from erroneous assumptions in the learning algorithm.

High bias can cause an algorithm to miss the relevant relations between features and target outputs (**underfitting**).



$$\theta_0 + \theta_1 x$$

Bias
(Underfitting)