Advanced Algorithms

Floriano Zini

Free University of Bozen-Bolzano
Faculty of Computer Science

Academic Year 2013-2014

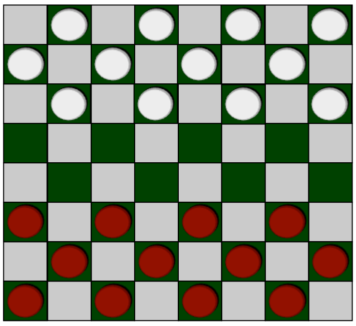
Lecture 11 – Linear regression

These slides are taken from **Andrew Ng**, **Machine Learning** on **Coursera** - https://class.coursera.org/ml-003/lecture/preview

Machine Learning definition

Arthur Samuel (1959)

- Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed



The Samuel Checkers-playing Program appears to be the world's first self-learning program

Machine Learning definition

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**

"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."

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task **T** in this setting?

- ❑ Classifying emails as spam or not spam
- ❑ Watching you label emails as spam or not spam.
- ❑ The number (or fraction) of emails correctly classified as spam/not spam.
- ❑ None of the above—this is not a machine 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."

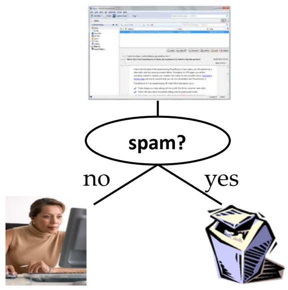
Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task **T** in this setting?

- **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.
- ❑ None of the above—this is not a machine learning problem

Examples of ML applications (1)

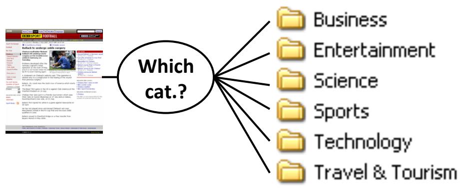
Email spam filtering

- **T**: to predict which email are spam for a given user
- **P**: % of emails correctly predicted
- **E**: a set of emails identified as spam/non spam for the user



Web pages categorization

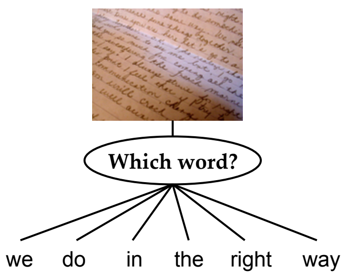
- **T**: to categorize Web pages in predefined categories
- **P**: % of Web pages correctly categorized
- **E**: a set of Web pages with specified categories



Examples of ML applications (2)

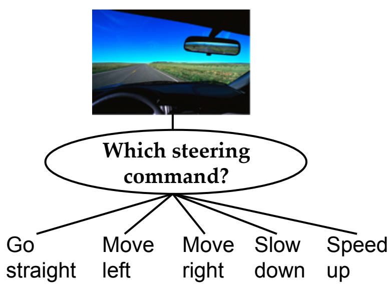
Handwriting recognition

- **T**: to recognize and classify handwritten words within images
- **P**: % of words correctly classified
- **E**: a database of handwritten words with given classifications (i.e., labels)



Robot driving

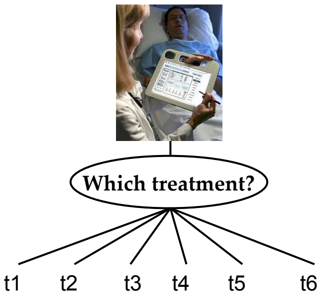
- **T**: to drive on public highways using vision sensors
- **P**: average distance traveled before an error (as judged by human observer)
- **E**: a sequence of images and steering commands recorded while observing a human driver



Examples of ML applications (3)

Medical diagnosis

- **T**: suggest treatments to doctors
- **P**: % of suggestions accepted by the doctors
- **E**: a database of electronic health records including <observations,treatment> pairs



Movie recommendation

- **T**: to recommend people movies they like
- **P**: overall satisfaction of the users
- **E**: movie ratings given by the users of the system

Machine learning algorithms

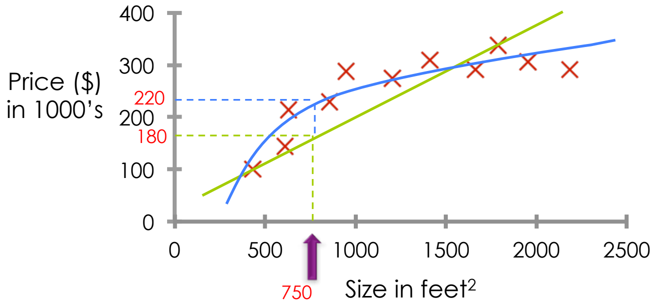
- Supervised learning
- Unsupervised learning (e.g., clustering)

- In this lecture 1 algorithm for supervised learning
  - Linear regression

- These topics are expanded in the courses:
  - Data Mining by Dr. Mouna Kacimi (second semester)
  - Information Search and Retrieval by Prof. Francesco Ricci (second semester)

Supervised learning

Housing price prediction

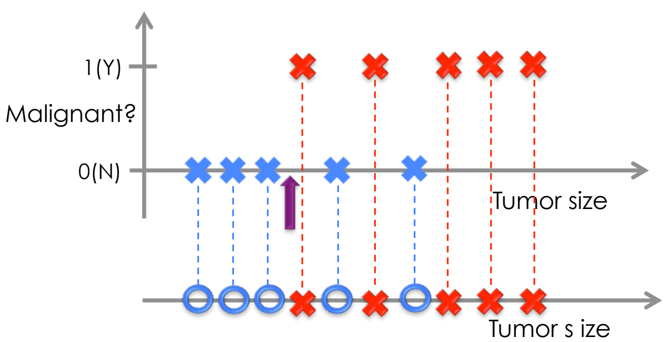


Supervised Learning
"right answers" given

Regression: Predict continuous valued output (price)

Supervised learning

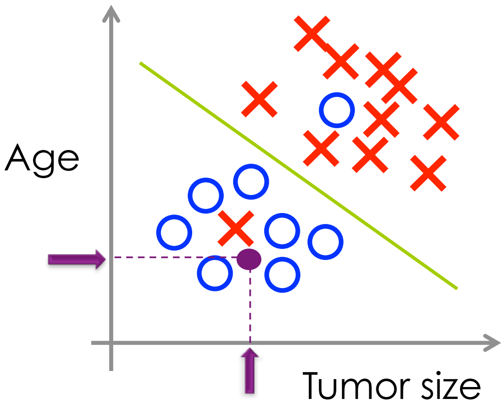
Breast cancer (malignant, benign)



Classification
Discrete valued output (0 or 1)

In this learning problem we consider just one ***feature***, i.e., the tumor size

Supervised learning



In this learning problem we consider two ***features***, i.e., the tumor size, and the age

In general, in real cases there are much more features. E.g.,

- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape
- …

You're running a company, and you want to develop learning algorithms to address each of two problems

- Problem 1: You have a large inventory of identical items. You want to predict how many of these items will sell over the next 3 months
- Problem 2: You'd like software to examine individual customer accounts, and for each account decide if it has been hacked/ compromised

Should you treat these as classification or as regression problems?

- Treat both as classification problems
- Treat problem 1 as a classification problem, problem 2 as a regression problem
- Treat problem 1 as a regression problem, problem 2 as a classification problem
- Treat both as regression problems

You're running a company, and you want to develop learning algorithms to address each of two problems

- Problem 1: You have a large inventory of identical items. You want to predict how many of these items will sell over the next 3 months
- Problem 2: You'd like software to examine individual customer accounts, and for each account decide if it has been hacked/ compromised

Should you treat these as classification or as regression problems?

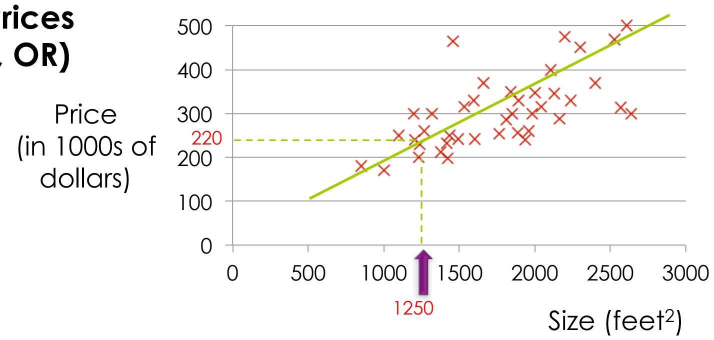
- ☐ Treat both as classification problems
- ☐ Treat problem 1 as a classification problem, problem 2 as a regression problem
- ✔ Treat problem 1 as a regression problem, problem 2 as a classification problem
- ☐ Treat both as regression problems

- 0 - not hacked
- 1 - hacked

Linear regression with one variable

Model representation

Housing Prices (Portland, OR)



<u>Supervised Learning</u>

Given the "right answer" for each example in the data

<u>Regression Problem</u>

Predict real-valued output

Model representation

Training set of housing prices (Portland, OR)

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

m=47

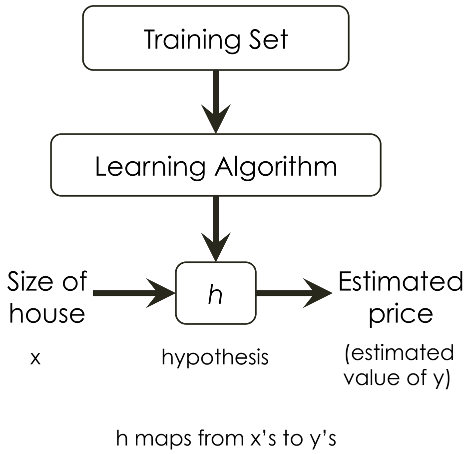
Notation:

- **m** = Number of training examples
- **x**'s = "input" variable / features
- **y**'s = "output" variable / "target" variable
- (**x, y**) = one training example
- ($\mathbf{x}^{(i)}, \mathbf{y}^{(i)}$)= i^{th} training example

E.g.,

- $x^{(1)}$=2104
- $x^{(2)}$=1416
- $y^{(1)}$=460
- ($x^{(3)}, y^{(3)}$) = (1534, 315)

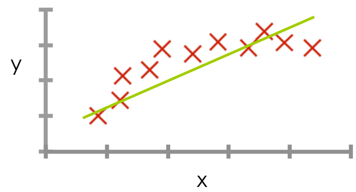
Model representation



How do we represent *h* ?

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

shorthand: h(x)



Linear regression with one variable.
Univariate linear regression.

Cost function

Training set

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

m=47

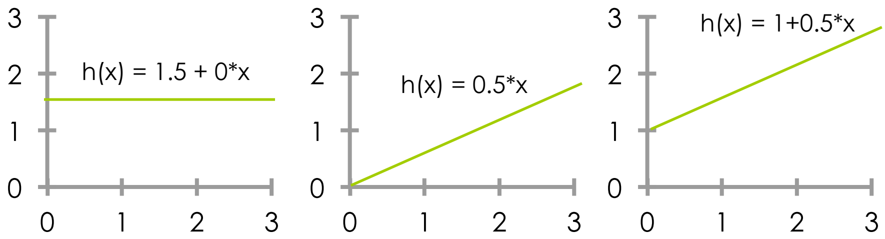
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

θ_i's: Parameters

How to choose θ_i's ?

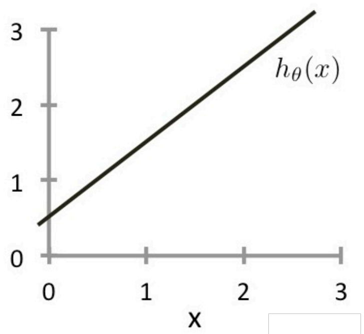
Cost function

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



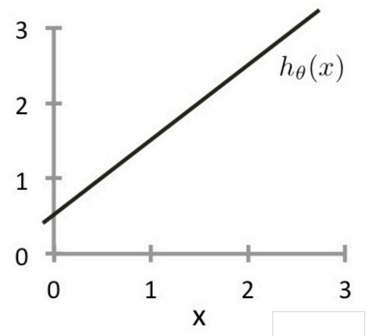
$\theta_0 = 1.5$, $\theta_1 = 0$ | $\theta_0 = 0$, $\theta_1 = 0.5$ | $\theta_0 = 1$, $\theta_1 = 0.5$

Consider the plot below of $h_\theta(x) = \theta_0 + \theta_1 x$. What are θ_0 and θ_1?



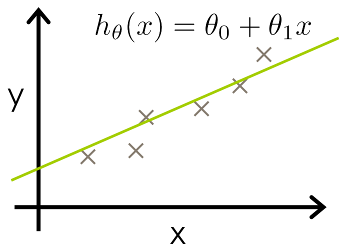
- $\theta_0 = 0, \theta_1 = 1$
- $\theta_0 = 0.5, \theta_1 = 1$
- $\theta_0 = 1, \theta_1 = 0.5$
- $\theta_0 = 1, \theta_1 = 1$

Consider the plot below of $h_\theta(x) = \theta_0 + \theta_1 x$. What are θ_0 and θ_1?



- ○ $\theta_0 = 0, \theta_1 = 1$
- ✔ $\theta_0 = 0.5, \theta_1 = 1$
- ○ $\theta_0 = 1, \theta_1 = 0.5$
- ○ $\theta_0 = 1, \theta_1 = 1$

Cost function



Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

Cost function

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

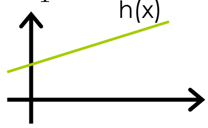
Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

$J(\theta_0, \theta_1)$ is also called squared error function

Cost function intuition 1

Hypothesis:

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



Parameters:

θ_0, θ_1

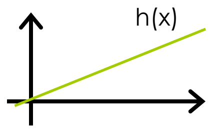
Cost Function:

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

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Simplified

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



θ_1

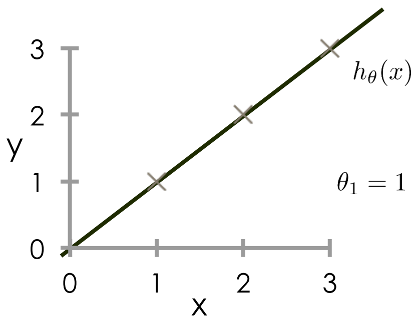
$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$

Cost function intuition 1

$h_\theta(x)$

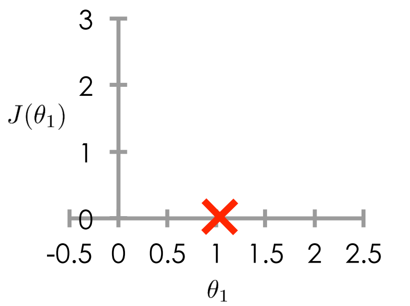
(for fixed θ_1, this is a function of x)



$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = 0$

$J(\theta_1)$

(function of the parameter θ_1)



J(1)= 0

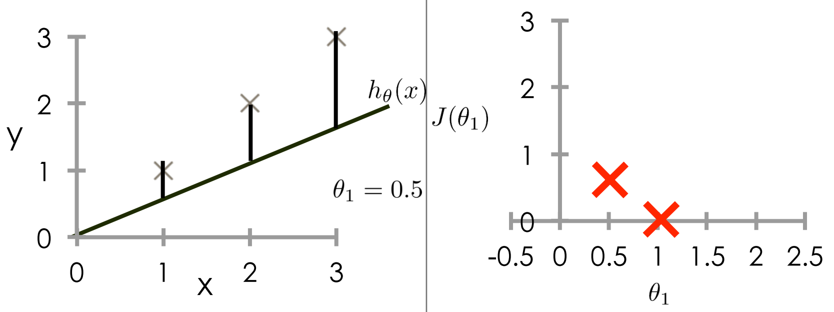
Cost function intuition 1

$h_\theta(x)$

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

$J(\theta_1)$

(function of the parameter θ_1)



$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

= 1/(2*3) * [(0.5-1)^2+(1-2)^2+(1.5-3)^2] ≈ 0.58

J(0.5)= 0.58

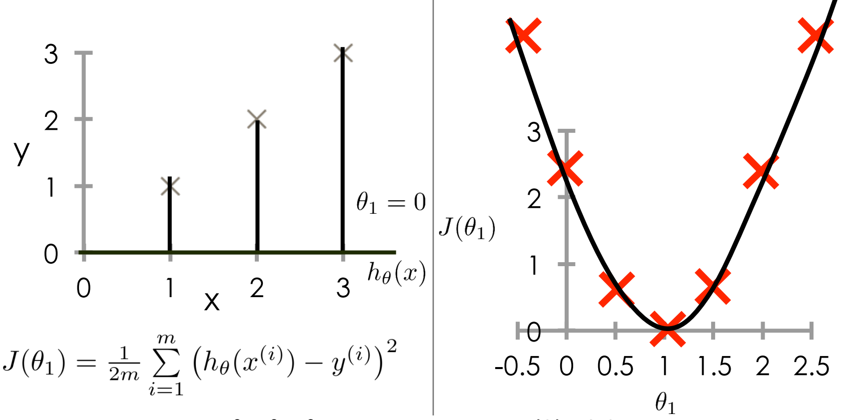
Cost function intuition 1

$h_\theta(x)$

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

$J(\theta_1)$

(function of the parameter θ_1)



$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

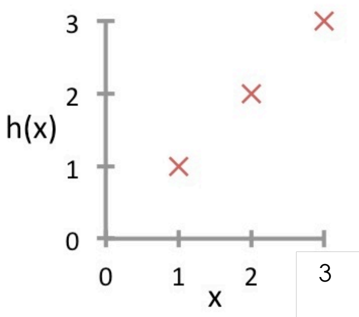
= 1/(2*3) * [1^2+2^2+3^2] ≈ 2.3

J(0)= 2.3

Suppose we have a training set with m=3 examples, plotted below. Our hypothesis representation is $h_\theta(x) = \theta_1 x$, with parameter θ_1. The cost function $J(\theta_1)$ is $J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$.

What is $J(0)$?

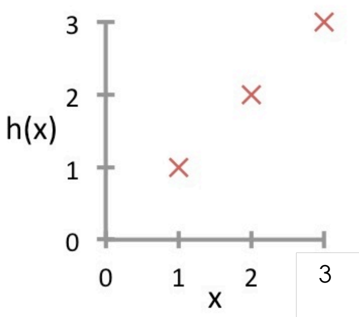
- ○ 0
- ○ 1/6
- ○ 1
- ○ 14/6



Suppose we have a training set with m=3 examples, plotted below. Our hypothesis representation is $h_\theta(x) = \theta_1 x$, with parameter θ_1. The cost function $J(\theta_1)$ is $J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$.

What is $J(0)$?

- ○ 0
- ○ 1/6
- ○ 1
- ✔ 14/6



Cost function intuition 2

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

Parameters: θ_0, θ_1

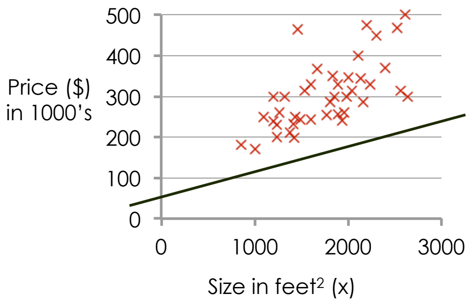
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Cost function intuition 2

$h_\theta(x)$

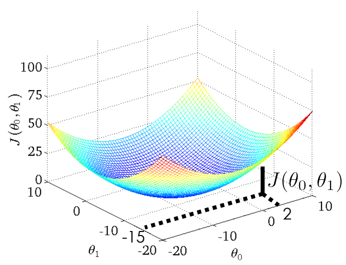
(for fixed θ_0, θ_1 , this is a function of x)



$h_\theta(x) = 50 + 0.06x$

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Cost function intuition 2

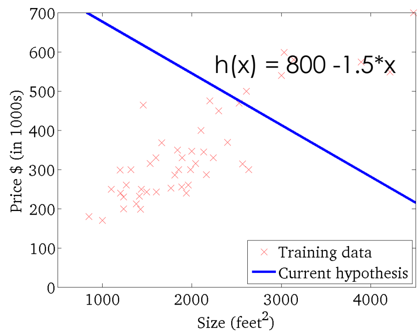
$h_\theta(x)$

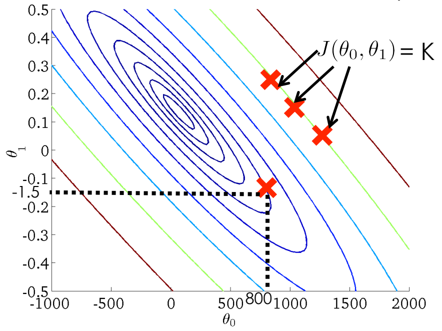
(for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

Contour plot





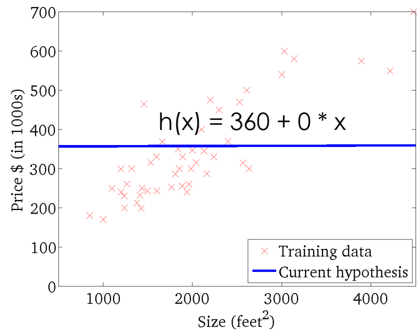
Cost function intuition 2

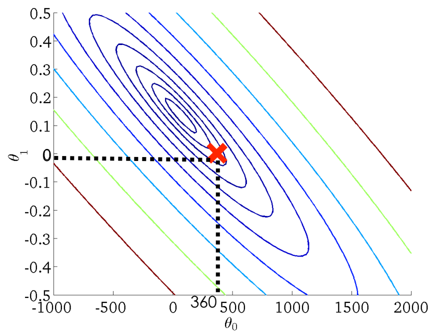
$h_\theta(x)$

(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)





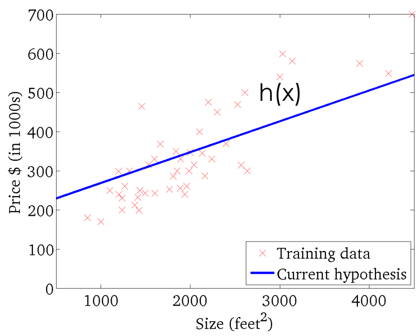
Cost function intuition 2

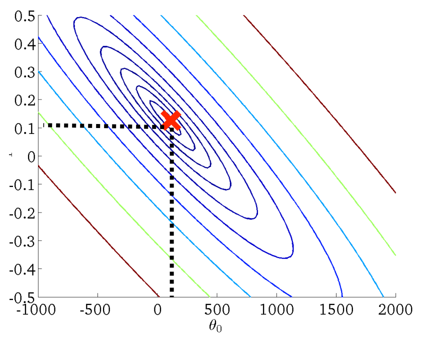
$h_\theta(x)$

(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)





Gradient descent

Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

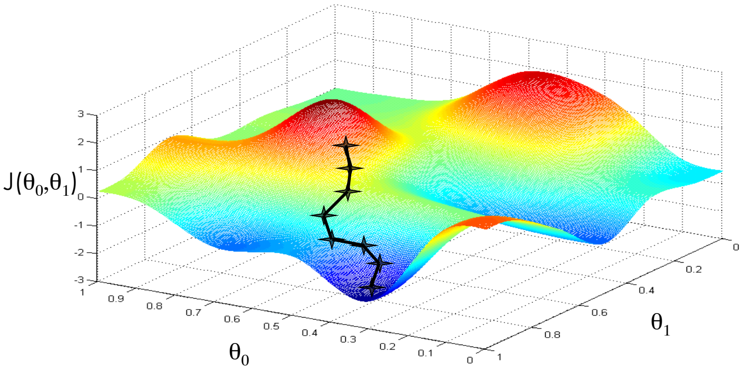
More in general:

- $J(\theta_0, \theta_1, \ldots, \theta_n)$
- $\min_\theta J(\theta_0, \theta_1, \ldots, \theta_n)$

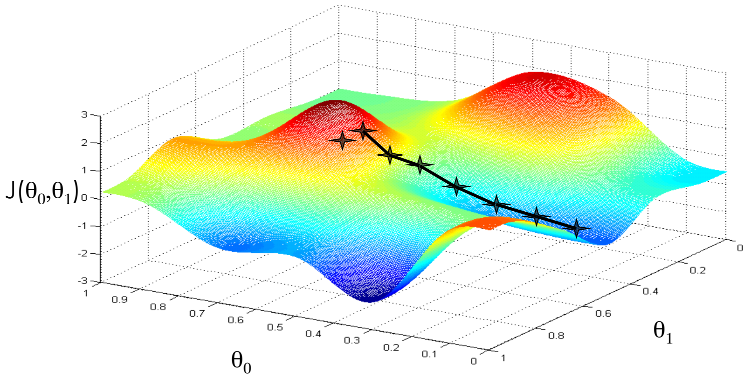
Outline:

- Start with some θ_0, θ_1 (e.g., $\theta_0 = 0$, $\theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum

Gradient descent



Gradient descent



Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

Learning rate

Correct: Simultaneous update

$$\begin{aligned} \text{temp0} &:= \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \text{temp1} &:= \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_0 &:= \text{temp0} \\ \theta_1 &:= \text{temp1} \end{aligned}$$

Incorrect:

$$\begin{aligned} \text{temp0} &:= \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \theta_0 &:= \text{temp0} \\ \text{temp1} &:= \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_1 &:= \text{temp1} \end{aligned}$$

Suppose $\theta_0 = 1, \theta_1 = 2$, and we simultaneously update θ_0 and θ_1 using the rule:

$\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$ (for j = 0 and j=1)

What are the resulting values of θ_0 and θ_1?

- ○ $\theta_0 = 1, \theta_1 = 2$
- ○ $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{2}$
- ○ $\theta_0 = 2 + \sqrt{2}, \theta_1 = 1 + \sqrt{2}$
- ○ $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{(1 + \sqrt{2}) \cdot 2}$

Suppose $\theta_0 = 1, \theta_1 = 2$, and we simultaneously update θ_0 and θ_1 using the rule:

$\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$ (for j = 0 and j=1)

What are the resulting values of θ_0 and θ_1?

- ○ $\theta_0 = 1, \theta_1 = 2$
- ✔ $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{2}$
- ○ $\theta_0 = 2 + \sqrt{2}, \theta_1 = 1 + \sqrt{2}$
- ○ $\theta_0 = 1 + \sqrt{2}, \theta_1 = 2 + \sqrt{(1 + \sqrt{2}) \cdot 2}$

Gradient descent intuition

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \underbrace{\alpha}_{\text{Learning rate}} \underbrace{\boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}}_{\text{Derivative}}$$

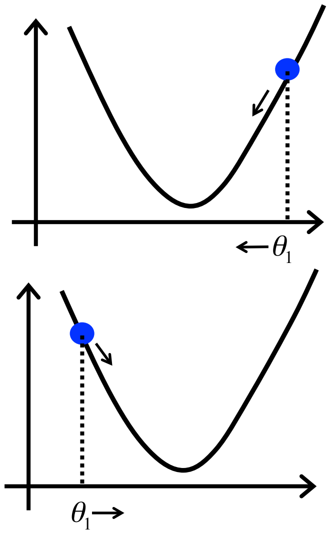
(simultaneously update $j = 0$ and $j = 1$)

}

Simplification

$$\min_{\theta_1} J(\theta_1), \theta_1 \in \Re$$

Gradient descent intuition



$J(\theta_1), \theta_1 \in \Re$

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)}$$

> 0

$$\theta_1 := \theta_1 - \alpha * (\text{positive number})$$

$$\frac{d}{d\theta_1} J(\theta_1) \leq 0$$

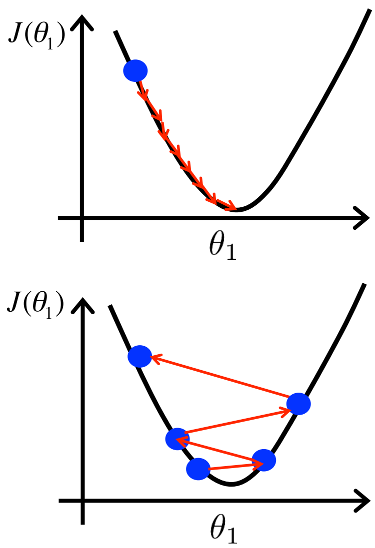
$$\theta_1 := \theta_1 - \alpha * (\text{negative number})$$

Gradient descent intuition

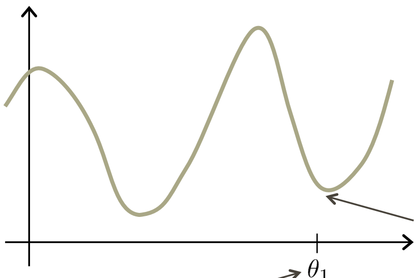
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow

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



What if θ_1 is already at the local minimum?



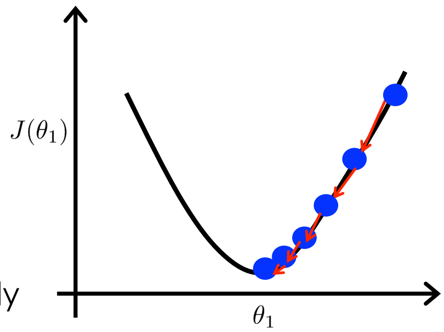
$$\theta_1 := \theta_1 - \alpha \underbrace{\boxed{\frac{d}{d\theta_1} J(\theta_1)}}_{= 0}$$

$$\theta_1 := \theta_1 - \alpha * 0$$ unchanged

Gradient descent intuition

Gradient descent can converge to a local minimum, even with the learning rate α fixed

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$



- As we approach a local minimum, gradient descent will automatically take smaller steps
- So, no need to decrease α over time

Gradient descent for linear regression

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$$

(for $j = 1$ and $j = 0$)

}

Key term

Linear Regression Model

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

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

We apply gradient descent to minimize $J(\theta_0,\theta_1)$: $\min_{\theta_0,\theta_1} J(\theta_0,\theta_1)$ $\theta_0,\theta_1 \in \Re$

Gradient descent for linear regression

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

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

Gradient descent for linear regression

Gradient descent algorithm

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

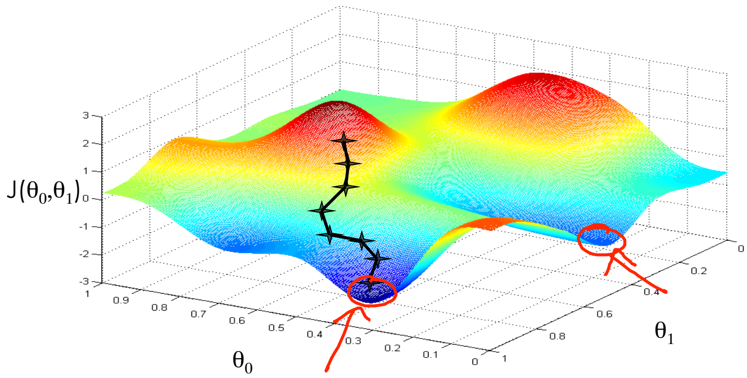
$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)} \\
&\quad \theta_1 := \theta_1 - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}} \\
&\}
\end{aligned}
$$

update θ_0 and θ_1 simultaneously

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

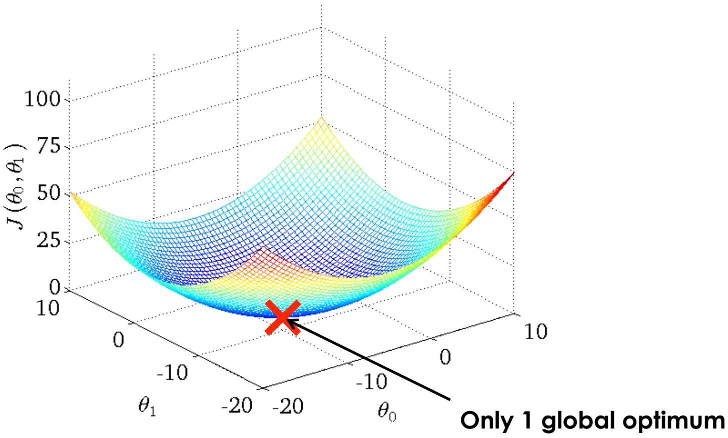
Gradient descent for linear regression

In general, the local optimum of the cost function ***J*** found by the gradient descent algorithm depends on the initialization of the parameters



Gradient descent for linear regression

The cost function ***J*** for liner regression is always a **convex** (or “bowl-shaped”) **function**



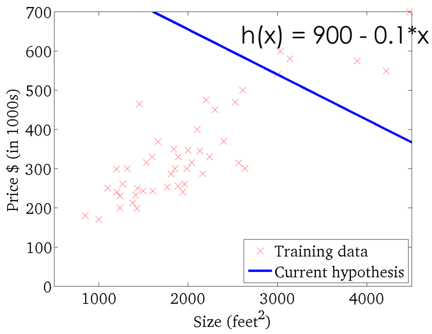
Gradient descent for linear regression

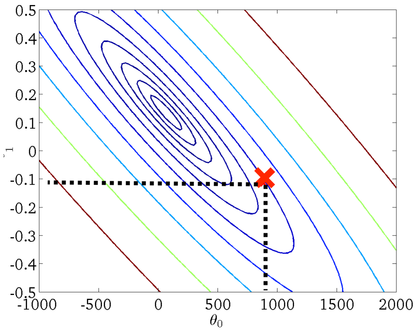
$h_\theta(x)$

(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)





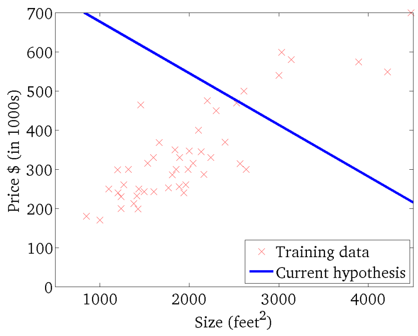
Gradient descent for linear regression

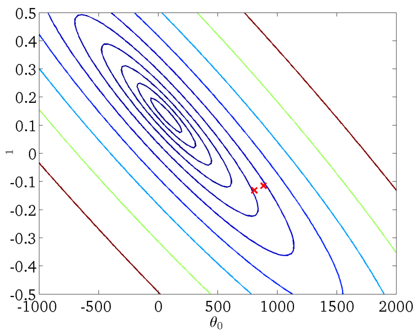
$h_\theta(x)$

(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)





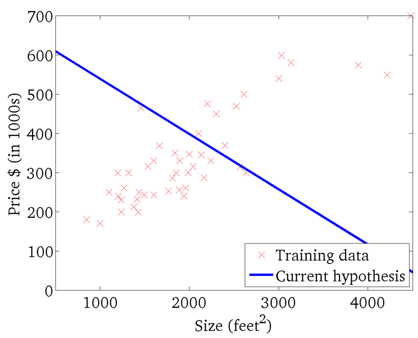
Gradient descent for linear regression

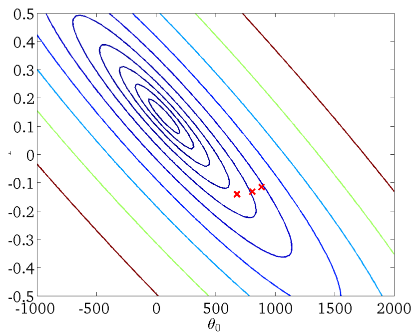
$h_\theta(x)$

(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)





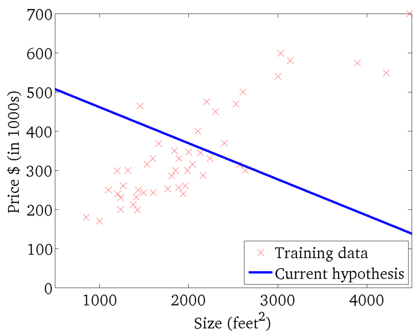
Gradient descent for linear regression

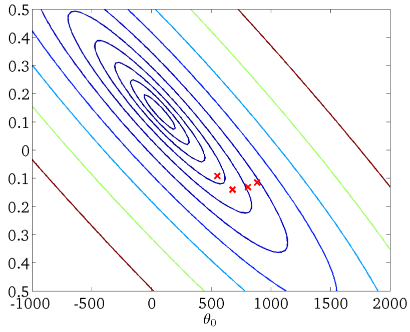
$h_\theta(x)$

(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)





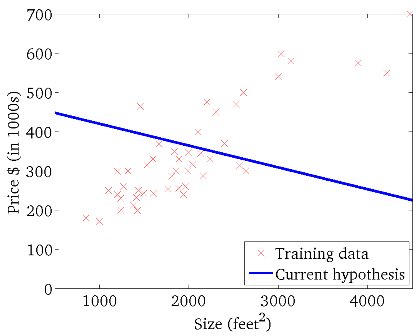
Gradient descent for linear regression

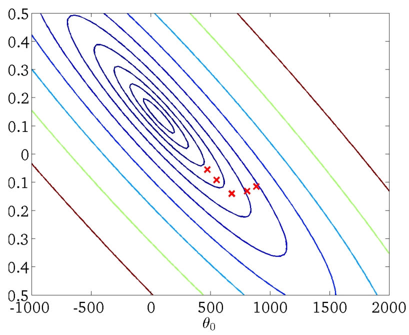
$h_\theta(x)$

(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)





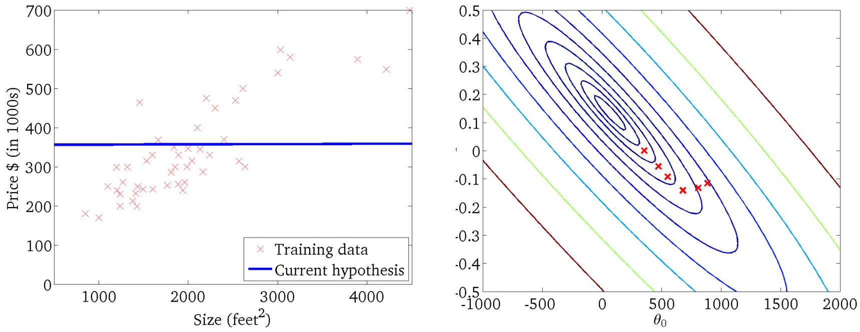
Gradient descent for linear regression

$h_\theta(x)$

(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)



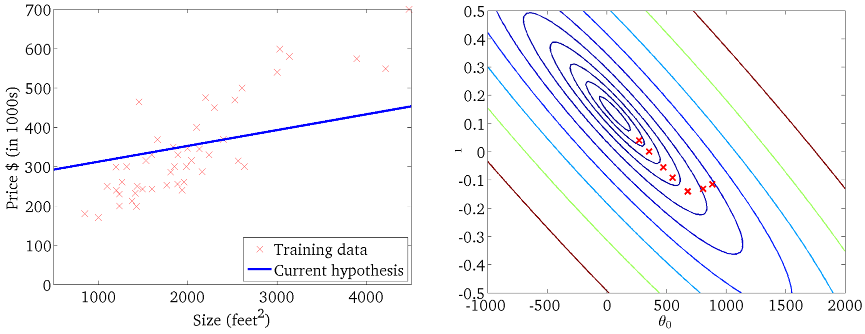
Gradient descent for linear regression

$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

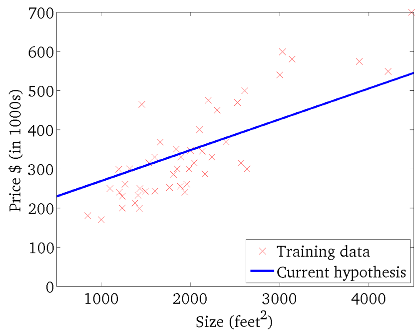
(function of the parameters θ_0, θ_1)



Gradient descent for linear regression

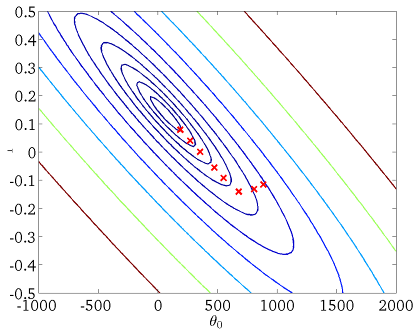
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

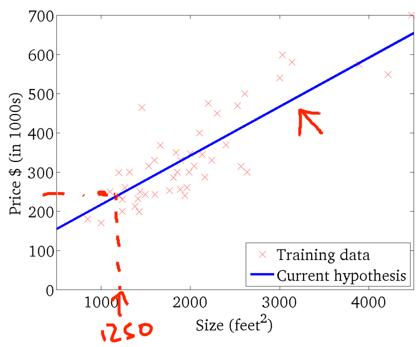
(function of the parameters θ_0, θ_1)



Gradient descent for linear regression

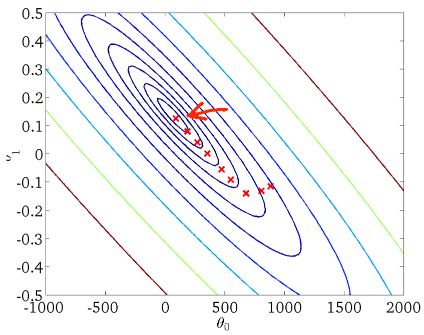
$h_\theta(x)$

(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)



Gradient descent for linear regression

"Batch" Gradient Descent

Each step of gradient descent uses **all** training examples in calculating the derivatives

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

There are variants:

- **Stochastic**: uses *1* example in each iteration
  - Faster algorithm, but may not reach the global minimum, even tough it gets close to it
- **Mini-batch**: uses *b* examples in each iteration, (*1<b<m*)

Which of the following are true statements? Select all that apply.

- [ ] To make gradient descent converge, we must slowly decrease α over time.
- [ ] Gradient descent is guaranteed to find the global minimum for any function $J(\theta_0, \theta_1)$.
- [ ] Gradient descent can converge even if α is kept fixed. (But α cannot be too large, or else it may fail to converge.)
- [ ] For the specific choice of cost function $J(\theta_0, \theta_1)$ used in linear regression, there are no local optima (other than the global optimum).

Which of the following are true statements? Select all that apply.

- [ ] To make gradient descent converge, we must slowly decrease α over time.
- [ ] Gradient descent is guaranteed to find the global minimum for any function $J(\theta_0, \theta_1)$.
- [x] Gradient descent can converge even if α is kept fixed. (But α cannot be too large, or else it may fail to converge.)
- [x] For the specific choice of cost function $J(\theta_0, \theta_1)$ used in linear regression, there are no local optima (other than the global optimum).

Linear regression with multiple variables

Multiple features

Single feature (variable)

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

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

Multiple features (variables)

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
…	…	…	…	…

Multiple features

Multiple features (variables)

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

m=47

n=4

Notation:

- **n** = number of features
- $\mathbf{x}^{(i)}$ = input (features) of **i**th training example
- $\mathbf{x}^{(i)}_j$ = value of feature **j** in **i**th training example

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix} \qquad x_3^{(2)} = 2$$

Multiple features

Hypothesis:

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

Now: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

E.g.: $h_\theta(x) = 80 + 0.1 * x_1 + 0.01 * x_2 + 3 * x_3 - 2 * x_4$

size # bedrooms # floors age

Multiple features

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$ $x_0^{(i)} = 1$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \Re^{n+1} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \Re^{n+1} \quad h_\theta(x) = \theta_0 \underbrace{x_0}_{=1} + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T x$$

Multivariate linear regression

Gradient descent for multiple variables

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 \underset{=1}{x_0} + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\boxed{\theta_0, \theta_1, \ldots, \theta_n} \longrightarrow \theta \in \Re^{n+1}$ n+1 dimensional vector

Cost function:

$$\underbrace{\boxed{J(\theta_0, \theta_1, \ldots, \theta_n)}}_{J(\theta)} = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \overbrace{\boxed{J(\theta_0, \ldots, \theta_n)}}^{J(\theta)}$$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient descent for multiple variables

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

E.g., n=2

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$