Important Dates

- Assignment #1: Intro to R/Python, GitHub, and developing a Portfolio – Opens 9/13
- Research Proposal Instructions will be handed out. (9/13)
- Assignment #2: Applying Analysis Techniques and Statistical Inference to Data: Opens 10/4
- Assignment #1: Due 10/4
- Students Research Proposals Due: (10/4)

Agenda

- **Linear Algebra Review**
- **How NASA Finds Critical Data Through a Knowledge Graphs**
  - GWU Elliott School, Room B-12
    1957 E St. NW
    Washington, DC
  - 6:30pm – 8pm
- Do HW 0 at home. Will review on next week.

Matrix: Rectangular array of numbers:

Dimension of matrix: number of rows x number of columns

Matrix Elements (entries of matrix)

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

$A_{ij} =$ "i, j entry" in the i^{th} row, j^{th} column.

Vector: An n x 1 matrix.

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$y_i = i^{th}$ element

1-indexed vs 0-indexed:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Matrix Addition

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \end{bmatrix} =$$

Scalar Multiplication

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} /4 =$$

Combination of Operands

$$3 \times \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix} /3$$

Example

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \end{bmatrix} =$$

Details:

$$A \times x = y$$

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \times \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

m x n matrix (m rows, n columns)	n x 1 matrix (n-dimensional vector)	m-dimensional vector

To get y_i, multiply A's i^{th} row with elements of vector x, and add them up.

Example

$$\begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -1 & -2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} =$$

House sizes:

2104
1416
1534
852

$$h_\theta(x) = -40 + 0.25x$$

Example

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} =$$

Details:

$$A \times B = C$$

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \times \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

m x n matrix (m rows, n columns)

n x o matrix (n rows, o columns)

m x o matrix

The i^{th} column of the matrix C is obtained by multiplying A with the i^{th} column of B. (for i = 1,2,…,o)

Example

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 3 & 2 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} =$$

House sizes:

$$2104$$
$$1416$$
$$1534$$
$$852$$

Have 3 competing hypotheses:

1. $h_\theta(x) = -40 + 0.25x$
2. $h_\theta(x) = 200 + 0.1x$
3. $h_\theta(x) = -150 + 0.4x$

Matrix

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times$$

Matrix

$$\begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

Let A and B be matrices. Then in general,

$A \times B \neq B \times A$. (not commutative.)

E.g. $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$

$\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix}$

$$A \times B \times C.$$

Let $D = B \times C$. Compute $A \times D$.

Let $E = A \times B$. Compute $E \times C$.

Identity Matrix

Denoted I (or $I_{n \times n}$).

Examples of identity matrices:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
2 x 2

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
3 x 3

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
4 x 4

For any matrix A,

$$A \cdot I = I \cdot A = A$$

Not all numbers have an inverse.

Matrix inverse:

If A is an m x m matrix, and if it has an inverse,

$$AA^{-1} = A^{-1}A = I.$$

Matrices that don't have an inverse are "singular" or "degenerate"

Matrix Transpose

Example: $A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}$ $A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$

Let A be an m x n matrix, and let $B = A^T$.

Then B is an n x m matrix, and

$$B_{ij} = A_{ji}.$$

Software Engineering - Collaboration

Collaboration

- We work together on projects, travel to workshops and conferences because we can pool our collective knowledge and experience to achieve more together than we can separately.

Collaboration

- Collaboration:
- Idea, process, and culture
- Formal approach to sharing information, ideas, software, and results
- Deceptively simple

Collaboration

- Communication with your team
- Tracking and monitoring the progress of your students
- Collecting and reporting research results
- Sharing tools and software you've developed
- Tracking changes made to your software
- Discussing problems and bugs encountered in software
- Reporting bugs to developers
- Responding to problems your users have reported

Preparing For The Future

- At a minimum:
- Nightly software builds
  - Up to hourly software builds during “sprints”
- Frequent patches
- Increasingly complicated project management
- More frequent delays and disruptions in communications

Dealing With Success

- IT industry:
- Created formal approach called “DevOps” we can adapt and implement to prepare for the future
- Created tools to help individuals and teams collaborate more efficiently

DevOps

- Both culture and process that emphasizes agility, flexibility, and responsiveness to users of our software
- Continuous builds
- Continuous integration
- Continuous feedback

DevOps

- Code:
  - Develop and review software
  - Continuous integration
- Build:
  - Status of builds
  - Version controls
  - Merging software
- Test:
  - Results-oriented objective metrics
- Package:
  - Artifact repositories
  - Pre-deployment staging

DevOps

- Release:
  - Automate releases
  - Approve and manage changes
- Configure:
  - “Infrastructure as Code”
    - Cloud computing
  - Automate configuration
- Monitor:
  - Performance monitoring
    - Simulation, reconstruction, and analysis time to run

Collaborative Tools

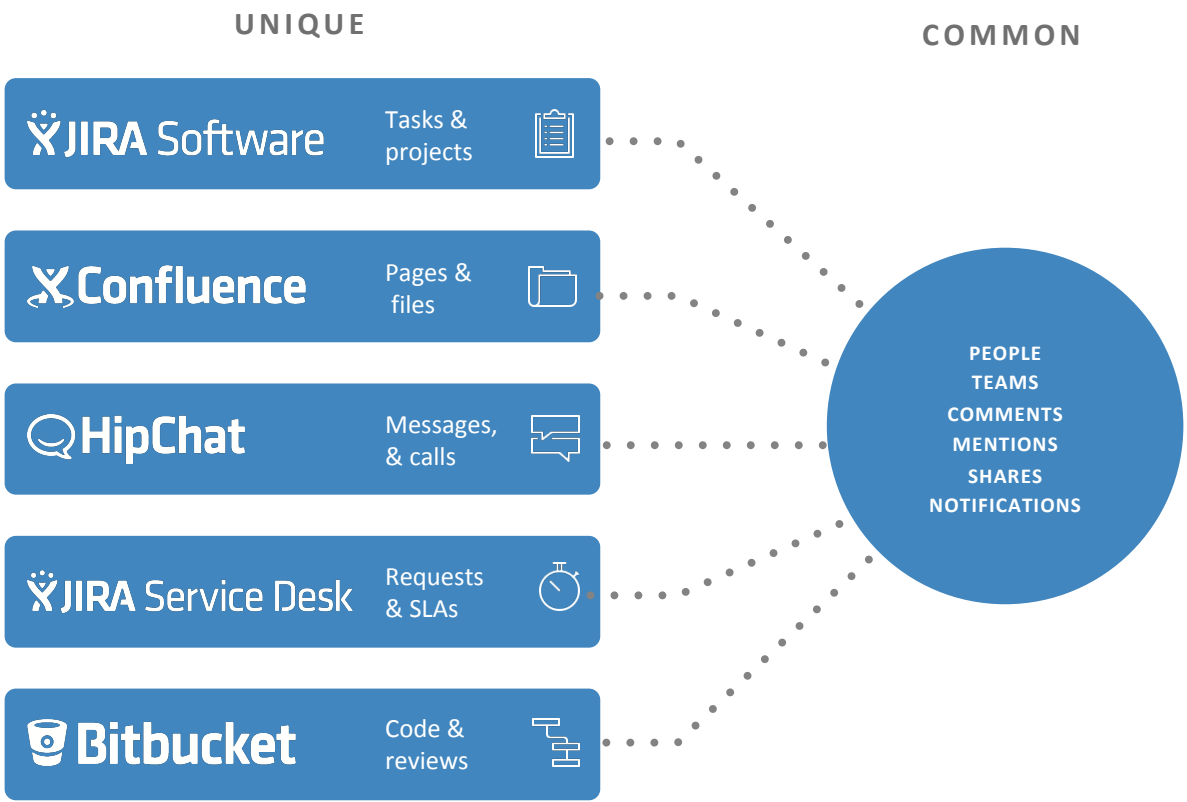
- DevOps relies on “toolchain”
- Interconnected software that supports the DevOps process and culture
- Tools support transparency, automation, and rapid dissemination of products

Collaborative Tools

- Many open-source or freeware tools exist
- No guarantee they will integrate successfully
- Slower releases, feature requests, and bugfixes
- Fewer add-ons to achieve specific tasks
- No single point of contact for support

The Atlassian Tool Suite

What Atlassian does for teams



Atlassian for software teams

Everything your team needs to build & deliver great software, fast

Plan & track

Adopt agile best practices. Plan projects, manage dependencies and track team progress.

Build & ship

Collaborate on code with inline comments and pull requests. Manage and share your Git and Mercurial repositories.

Collaborate & document

Keep teams connected with chat. Collaborate on product requirements, roadmaps and technical documentation.

Automate & deploy

Embrace continuous delivery. Automate builds, tests and releases in a single workflow.

What is a Programming Language?

- A ***programming language*** is a notational system for describing computation in machine-readable and human-readable form.
- Most of these forms are ***high-level languages***, which is the subject of the course.
- Assembly languages and other languages that are designed to more closely resemble the computer's instruction set than anything that is human-readable are ***low-level languages***.

Why Study Programming Languages?

- In 1969, Sammet listed 120 programming languages in common use – now there are many more!
- Most programmers never use more than a few.
  - Some limit their career's to just one or two.
- The gain is in learning about their underlying design concepts and how this affects their implementation.

The Six Primary Reasons

- Increased ability to express ideas
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of significance of implementation
- Better use of languages that are already known
- Overall advancement of computing

Programming Domains

- Scientific Applications
- Business Applications
- Artificial Intelligence
- Web Software

Artificial Intelligence

- Artificial Intelligence deals with emulating human-style reasoning on a computer.
- These applications usually involve symbolic computation, where most of the symbols are names and not numbers.
- The most common data structure is the list, not the matrix or array as in scientific computing and not the record as in business computing
- Artificial intelligence requires more flexibility than other programming domains.

Artificial Intelligence Languages

- The first AI language was IPL (International Processing Language, developed by the Rand Corporation. Its low-level design led to its limited use.
- John McCarthy of MIT developed LIST for the IBM 704 (which eventually led to Scheme and Common LISP). LISP is a recursion-oriented, list-processing language that facilitated game-playing programs.
- Yngve of MIT developed COMIT, a string-processing language, which was followed by AT&T's SNOBOL.
- Prolog was developed by Colmerauer, Roussel and Kowalski based on predicate calculus and mathematical logic.

Systems Languages

- Assembly languages were used for a very long time operating systems programming because of its power and efficiency.
- CPL, BCPL, C and C++ were later developed for this purpose.
- Other languages for systems programming included PL/I, BLISS, and extended ALGOL.

Web Software

- Eclectic collection of languages:
  - **Markup** (e.g., HTML) – used for annotating a document in a manner that can be distinguished from the text.
  - **Scripting** (e.g., PHP) - the language that enable the script to run these commands and typically include control structures such as if-then-else and while-do.
  - **General-purpose** (e.g., Java) – can be used for a wide range of programming jobs.

Language Evaluation Criteria

- **Readability** – the ease with which programs can be read and understood.
- **Writability** – the ease with which programs can be developed for a given program domain.
- **Reliability** – the extent to which a program will perform according to its specifications.

An Example of Multiplicity

- All of the following add one to the variable count in C:

```
count = count + 1;
count += 1;
count++;
++count;
```

Do they mean the same thing?

Data Types and Structures

- A more diverse set of data types and the ability of programmers to create their own increased program readability:
  - Booleans make programs more readable:

    `TimeOut = 1` vs. `TimeOut = True`
  - The use of records to store complex data objects makes programs more readable:

```
CHARACTER*30 NAME(100)
INTEGER AGE(100), EMPLOYEE_NUM(100)
REAL SALARY(100)
```

    Wouldn't it better if these were an array of records instead of 4 parallel arrays?

Syntax

- Most syntactic features in a programming language can enhance readability:
  - **Identifier forms** – older languages (like FORTRAN) restrict the length of identifiers, which become less meaningful
  - **Special words** – in addition to `while`, `do` and `for`, some languages use special words to close structures such as endif and endwhile.
  - **Form and meaning** – In C a `static` variable within a function and outside a function mean two different things – this is undesirable.

Contributing Factors To Reliability

- **Type Checking** – a large factor in program reliability. Compile-time type checking is more desireable. C's lack of parameter type checking leads to many reliability problems.
- **Exception Handling** – the ability to catch run-time errors and make corrections can prevent reliability problems.
- **Aliasing** – having two or more ways of referencing the same data object can cause unnecessary errors.

Cost of Use

- Cost of program execution
  - A slower program is more expensive to run on a slower computer.
  - In an era of faster, cheaper computer, this is less of a concern.
- Cost of program translation
  - Optimizing compilers are slower than some other compilers designed for student programs, which will not run as many times..
- Cost of program creation, testing and use
  - How quickly can you get the program executing ***correctly***.
- Cost of program maintenance
  - How expensive will it be to modify the program when changes are needed in subsequent years?

Influences on Language Design

Other factors have had a strong influence on programming language design:

- Computer Architecture
- Programming Methodologies

Computer Architecture

- Most computers are still based on the von Neumann architecture, which view memory as holding both instructions and data interchangably.
- This has influenced the development of imperative languages and has stifled the adaption of functional languages.
- As parallel processing computers are developed, there have been several attempts made to develop languages that exploit their features.

Programming Methodologies

- New methods of program development have led to advances in language design:
- These have included:
  - structured programming languages
  - data abstraction in object-oriented languages

Language Categories

- There are four different programming language paradigms:
  - Imperative
  - Functional
  - Declarative
  - Object-Oriented

Imperative Languages

- Imperative languages are command-driven or statement-oriented languages.
- The basic concept is the machine state (the set of all values for all memory locations).
- A program consists if a sequence of statements and the execution of each statement changes the machine state.
- Programs take the form:

```
statement1;
statement2;
  … …
```

- FORTRAN, COBOL, C, Pascal, PL/I are all imperative languages.

Functional Languages

- An functional programming language looks at the function that the program represents rather than the state changes as each statement is executed.
- The key question is: What function must be applied to our initial machine and our data to produce the final result?
- Statements take the form:

  $function_n(function_1, function_2, \ldots (data)) \ldots)$

- ML, Scheme and LISP are examples of functional languages.

A Function GCD in C++

```
//gcd() -  A version of greatest common
// divisor written in C++ in
// function style
int   gcd(int u, int v)
{
  if (v == 0)
   return(u);
  else
   return(v, u % v);
}
```

Rule-Based Languages

- Rule-based or ***declarative*** languages execute checking to see if a particular condition is true and if so, perform the appropriate actions.
- The enabling conditions are usually written in terms of predicate calculus and take the form:

  $condition_1 \rightarrow action_1$
  $condition_2 \rightarrow action_2$
  … … …
- Prolog is the best know example of a declarative language.

Object-Oriented Languages

- In object-oriented languages, data structures and algorithms support the abstraction of data and endeavor to allow the programmer to use data in a fashion that closely represents its real world use.
- Data abstraction is implemented by use of
  - ***Encapsulation*** – data and procedures belonging to a class can only be accessed by that classes (with noteworthy exceptions).
  - ***Polymorphism*** – the same functions and operators can mean different things depending on the parameters or operands,
  - ***Inheritance*** – New classes may be defined in terms of other, simpler classes.

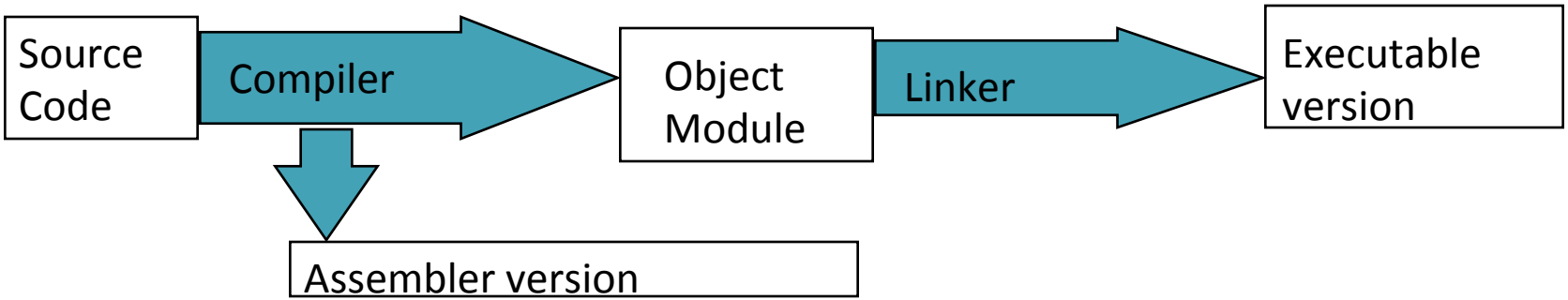
GCD in Java

```
public class IntWithGcd
{ public IntWithGcd( int val ){ value = val; }
  public int intValue() { return value; }
  public int gcd( int val );
  { int z= value;
    int y = v;
    while (y != 0)
    {
      int t = y;
      y = z % y;
      z = t;
    }
    return z;
  }
  private int value;
}
```

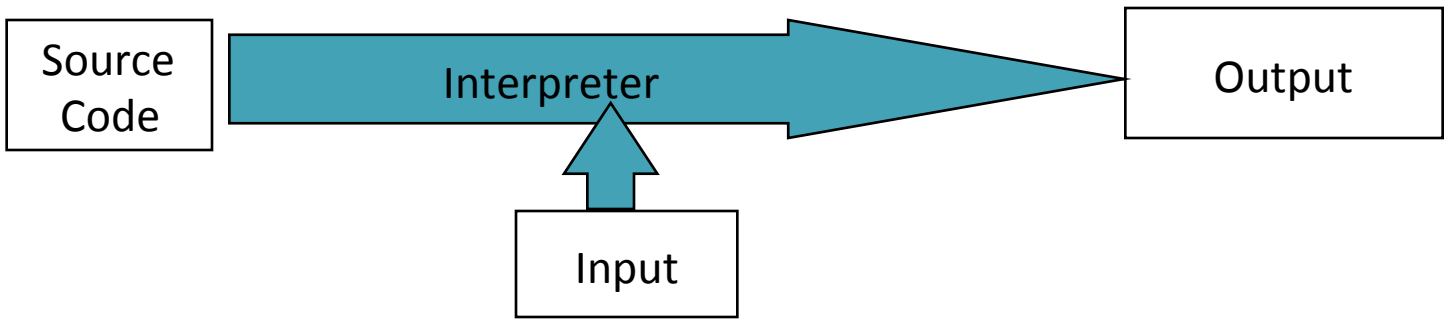
Implementation Methods

- Compilation
- Pure Interpretation
- Hybrid Implementation Systems

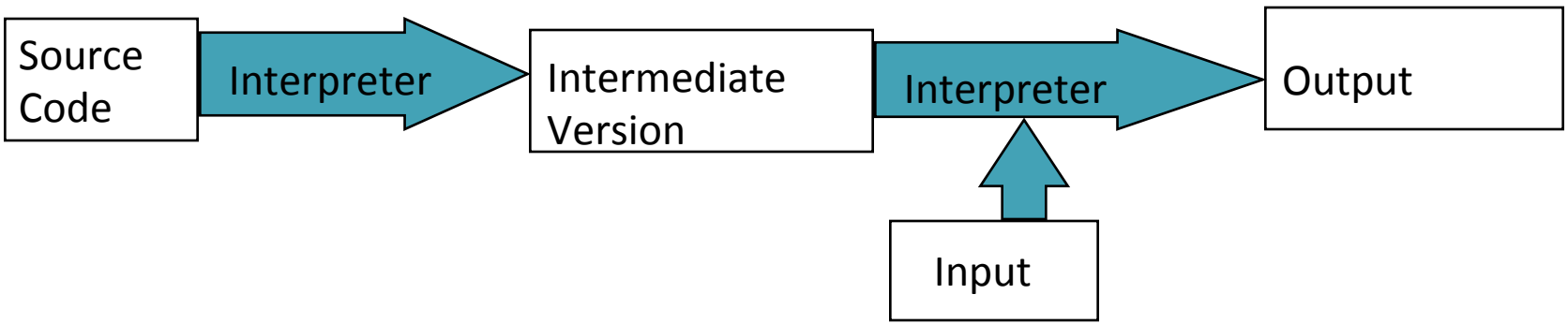
The Compiling Process



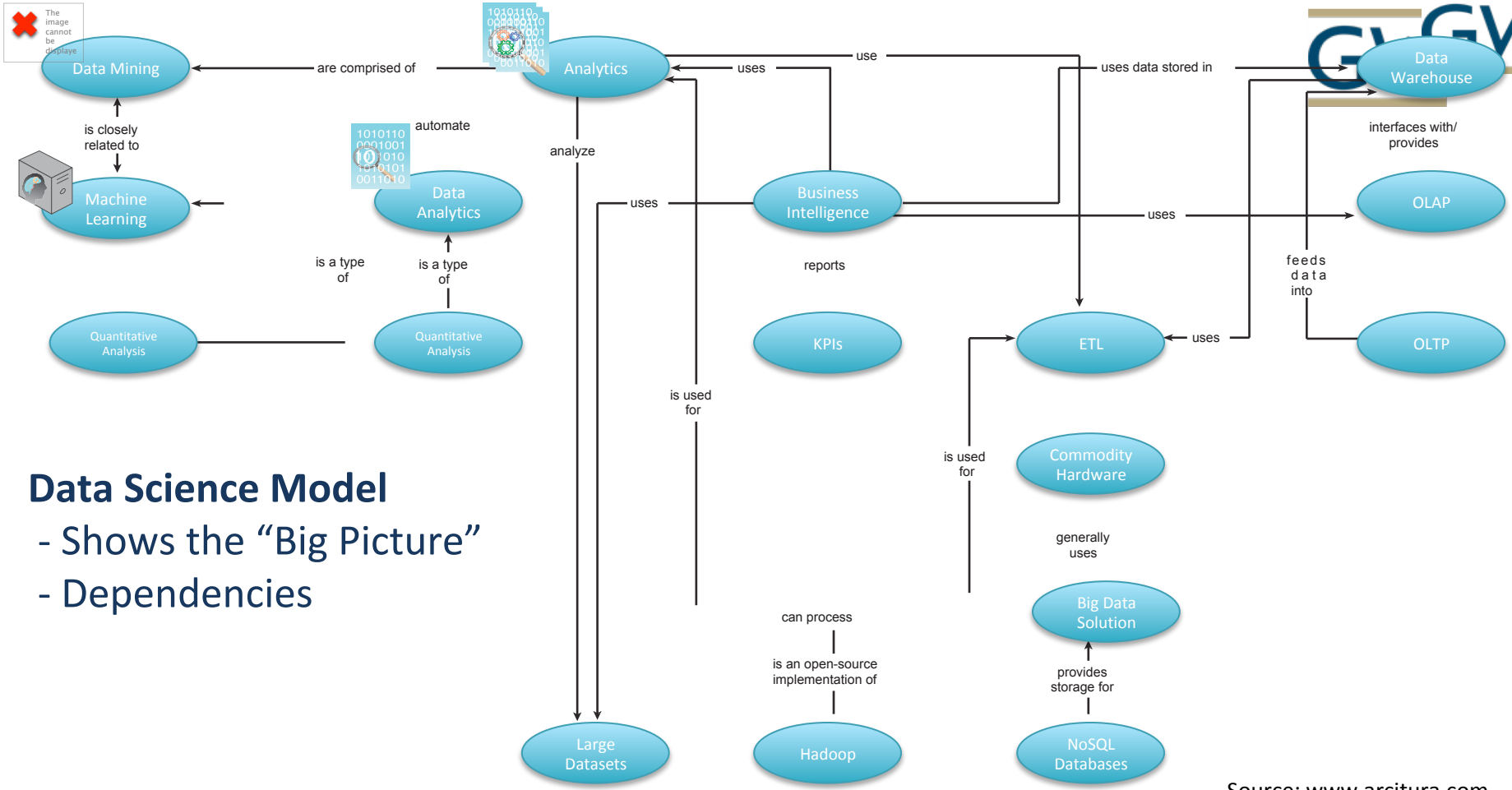
The Pure Interpretation Process



The Hybrid Interpretation Process



Data Science Architecture



Data Science Model

- Shows the "Big Picture"
- Dependencies

Source: www.arcitura.com

Benjamin Harvey, Ph.D., EMSE 6992 – Introduction to Data Analytics

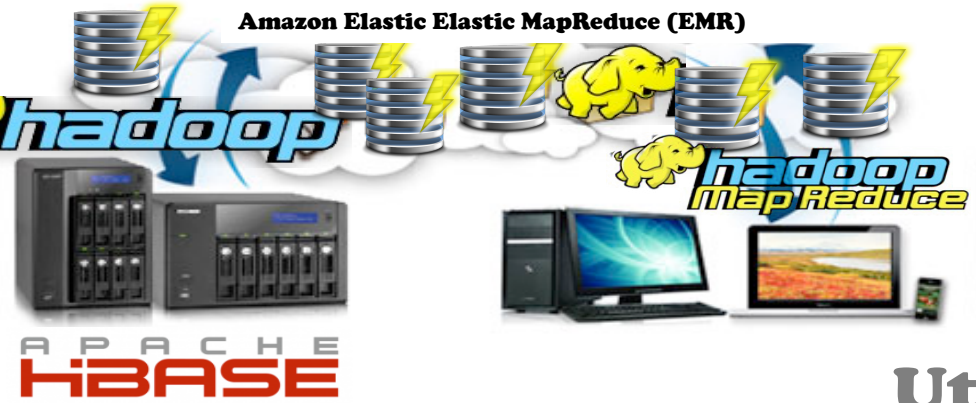
Cloud Infrastructures


DATA CLOUD
amazon web services™
STORAGE CLOUD
amazon web services™
GW
Amazon Elastic Elastic MapReduce (EMR)
hadoop
hadoop MapReduce
Amazon Simple Storage Service (S3)
APACHE HBASE
UTILITY CLOUD
amazon web services™
Amazon Elastic Compute Cloud (EC2)
CentOS
FreeBSD
suse
opensolaris

DATA CLOUD

amazon web services™

Amazon Elastic Elastic MapReduce (EMR)



APACHE HBASE

STORAGE CLOUD

amazon web services™

GW

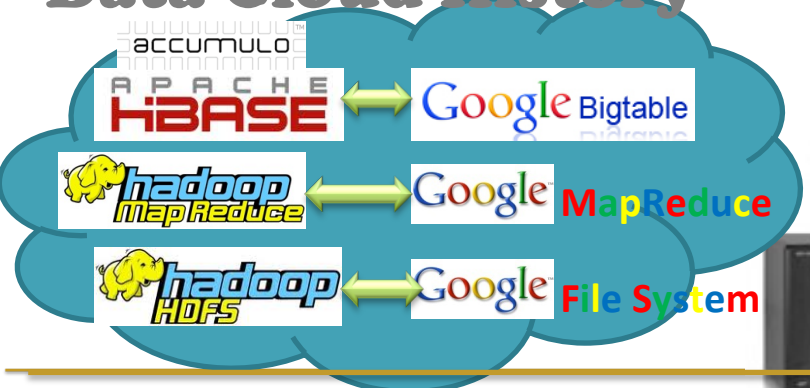
Amazon Simple Storage Service (S3)

Utility CLOUD

amazon web services™

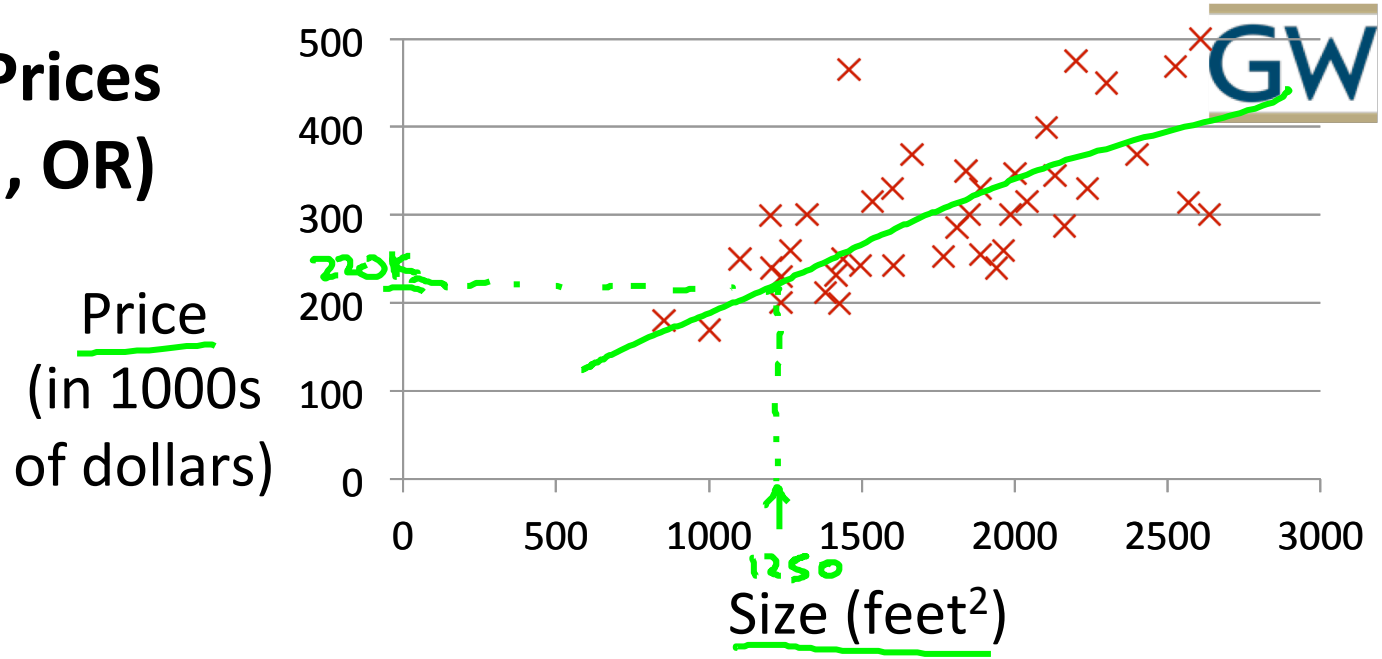
Amazon Elastic Compute Cloud (EC2)

Data Cloud History



REFERENCE SLIDES

Housing Prices (Portland, OR)



Supervised Learning

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

Regression Problem

Predict real-valued output

Classification: Discrete-valued output

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

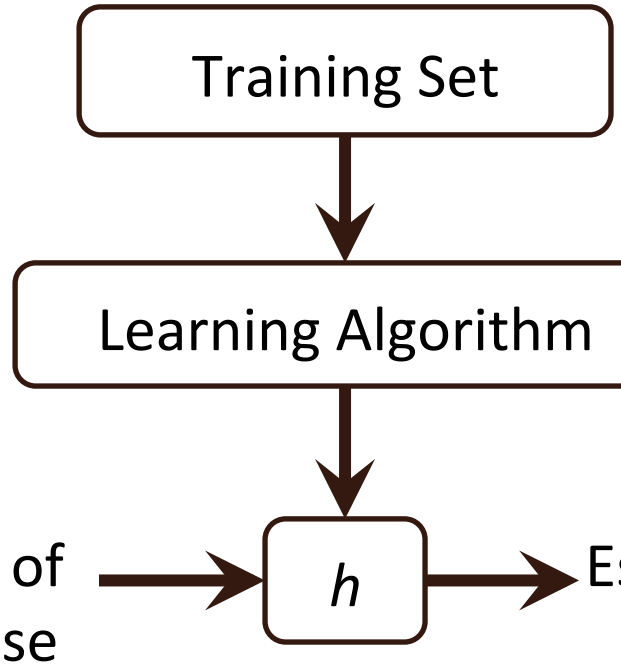
$x^{(1)} = 2104$

$x^{(2)} = 1416$

$y^{(1)} = 460$

(x, y) - one training example

$(x^{(i)}, y^{(i)})$ - i^{th} training example



x

hypothesis

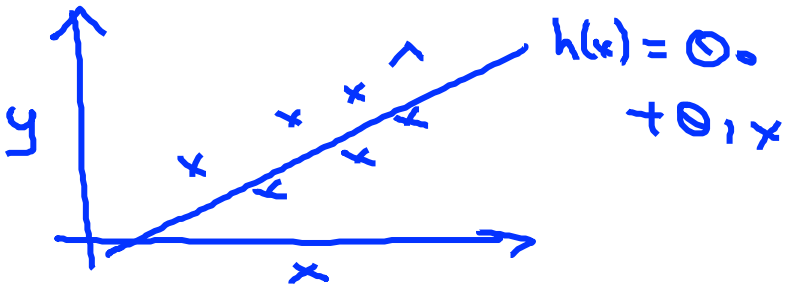
(estimated value of y)

h maps from x's to y's.

How do we represent h ?

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

Shorthand: $h(x)$



Linear regression with one variable. (x)

Univariate linear regression.

└ one variable

Training Set

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

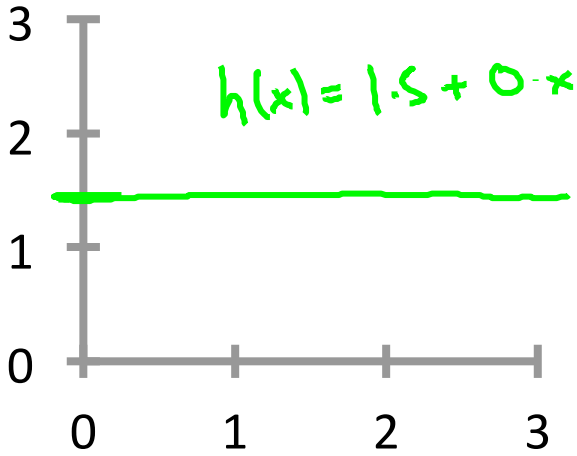
m = 47

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

θ_i's: Parameters

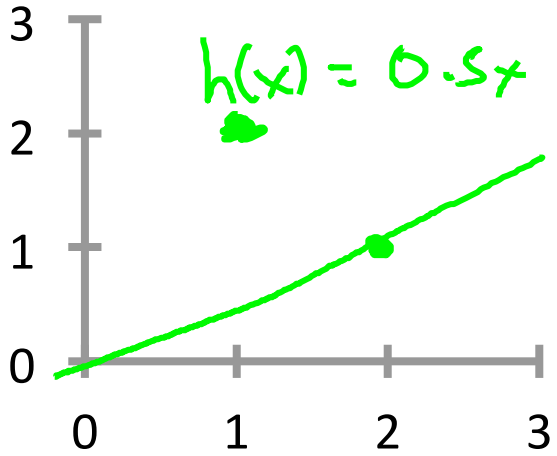
How to choose θ_i's ?

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



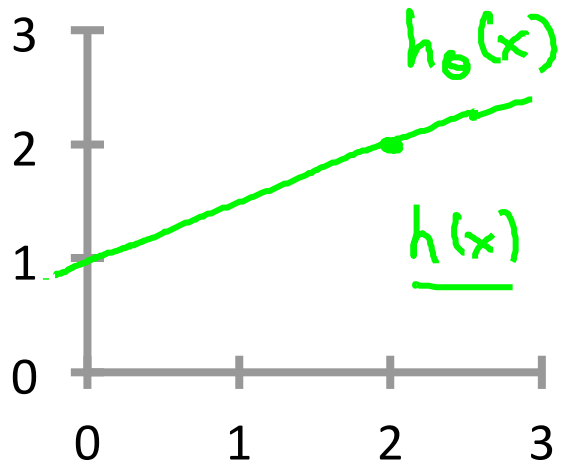
→ $\theta_0 = 1.5$

→ $\theta_1 = 0$



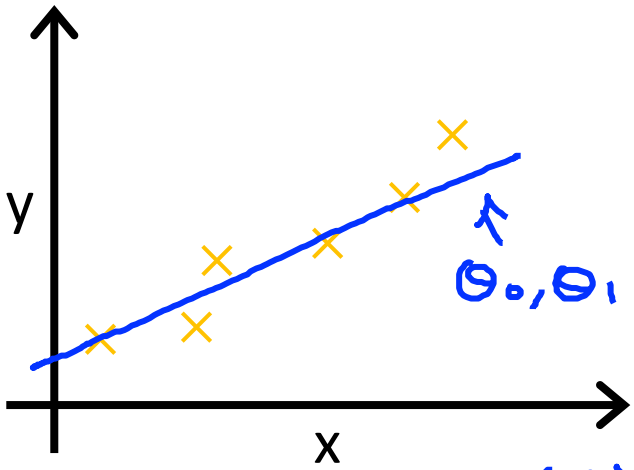
→ $\theta_0 = 0$

→ $\theta_1 = 0.5$



→ $\theta_0 = 1$

→ $\theta_1 = 0.5$



$(x^{(i)}, y^{(i)})$

#training examples

minimize θ_0, θ_1 $\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$

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

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

Minimize θ_0, θ_1 $J(\theta_0, \theta_1)$

Cost function

Squared error function

x, y

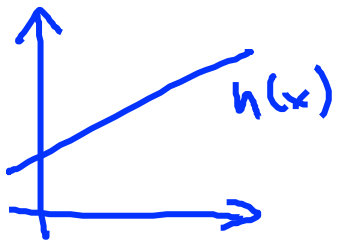
Simplified

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$



Cost Function:

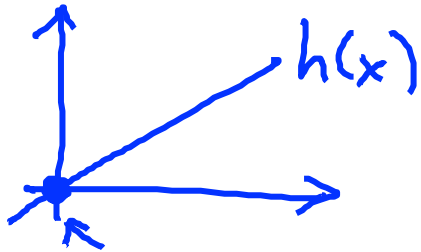
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

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

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

$\theta_0 = 0$

θ_1



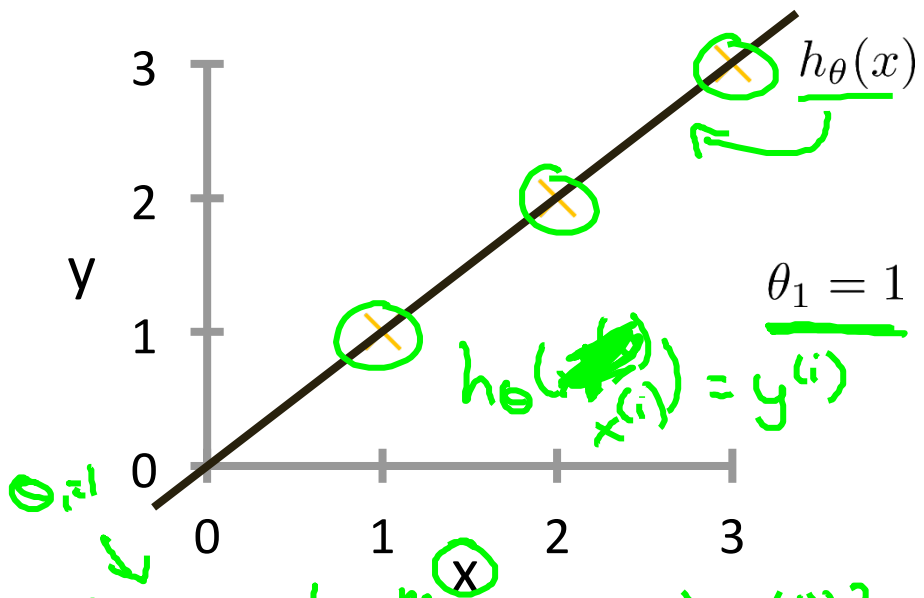
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$\theta_1 x^{(i)}$

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

→ $h_\theta(x)$

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



$h_\theta(x^{(i)}) = y^{(i)}$

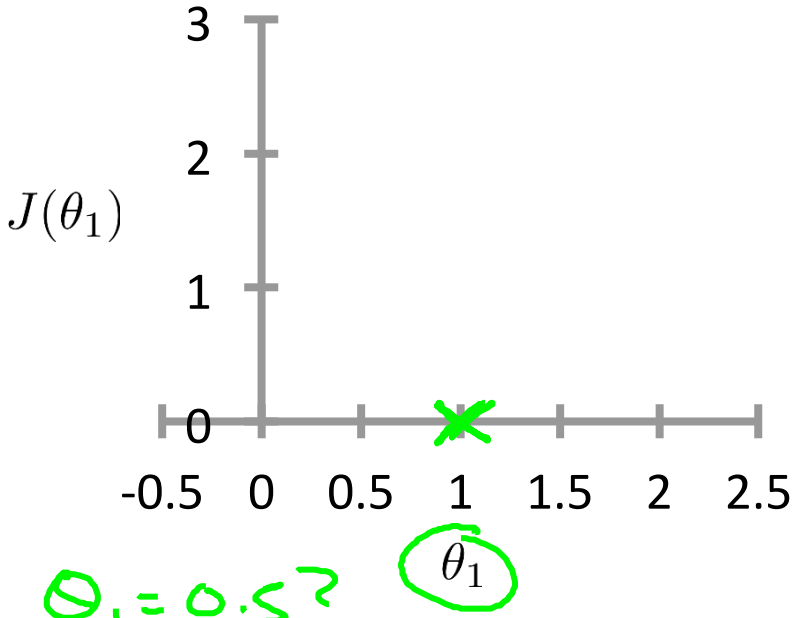
$\theta_1 = 1$

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

$= \frac{1}{2m} \sum_{i=1}^{m} (\theta_1 x^{(i)} - y^{(i)})^2 = \frac{1}{2m}(0^2 + 0^2 + 0^2) = 0^2$

→ $J(\theta_1)$

(function of the parameter θ_1)

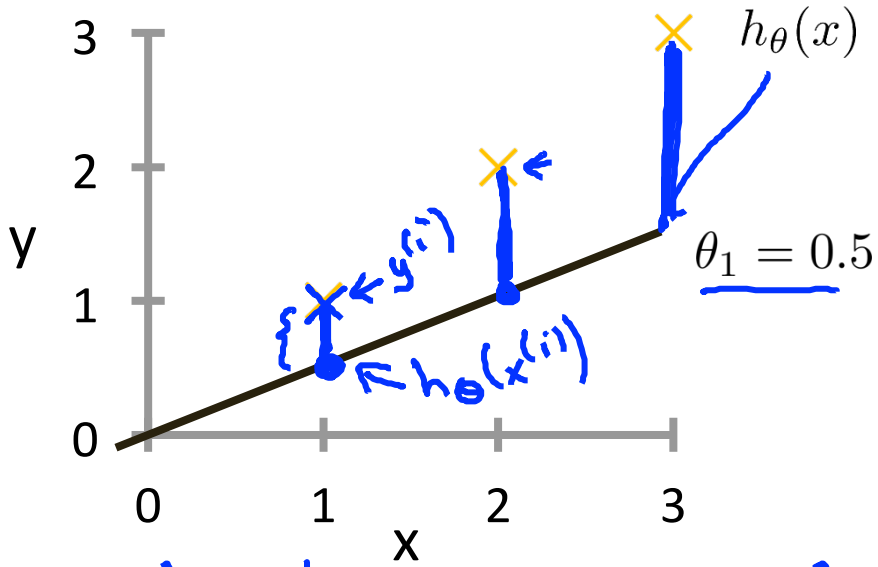


$\theta_1 = 0.5$?

$J(1) = 0$

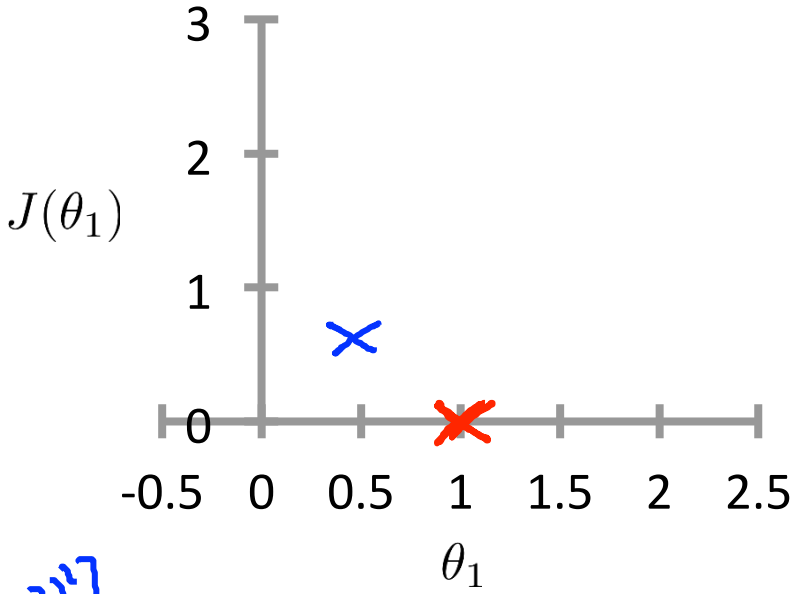
$h_\theta(x)$

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



$J(\theta_1)$

(function of the parameter θ_1)



$J(0.5) = \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$

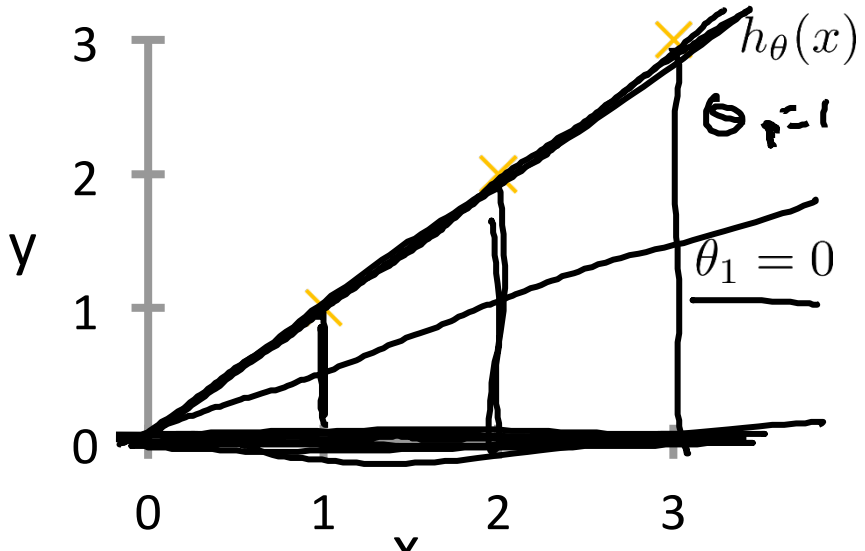
$= \frac{1}{2\times 3}(3.5) = \frac{3.5}{6} \approx 0.58$

$\theta_1 = 0?$

$J(0) = ?$

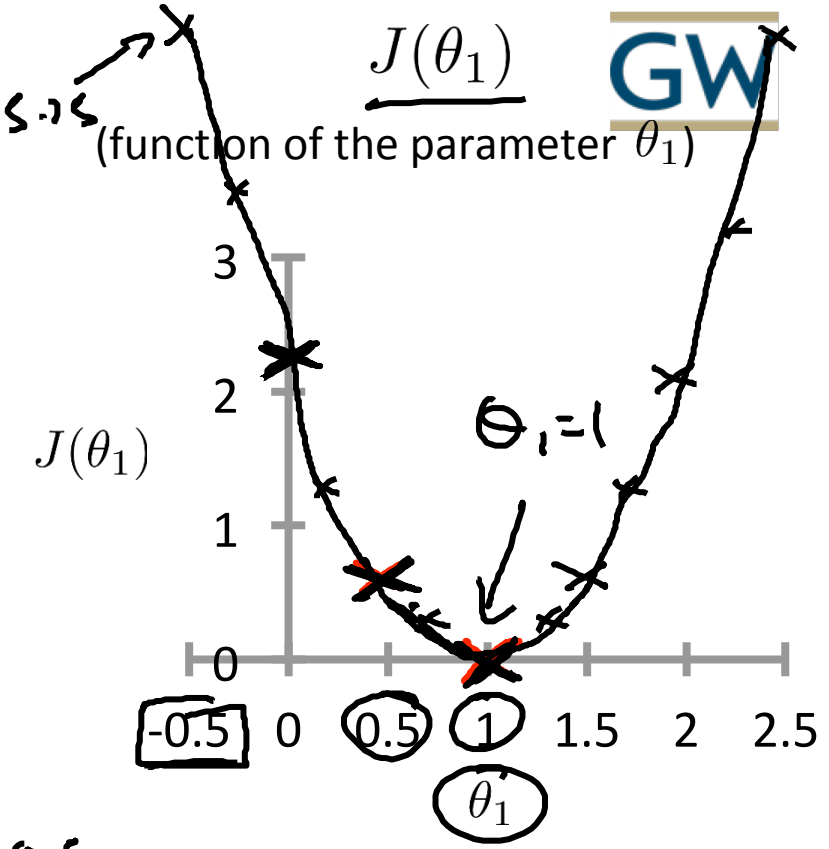
$h_\theta(x)$

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



$J(\theta_1)$

(function of the parameter θ_1)



$J(0) = \frac{1}{2m}(1^2 + 2^2 + 3^2)$

$= \frac{1}{6} \cdot 14 \approx 2.3$

$h(x) = -0.5x$

minimize $J(\theta_1)$
θ_1

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

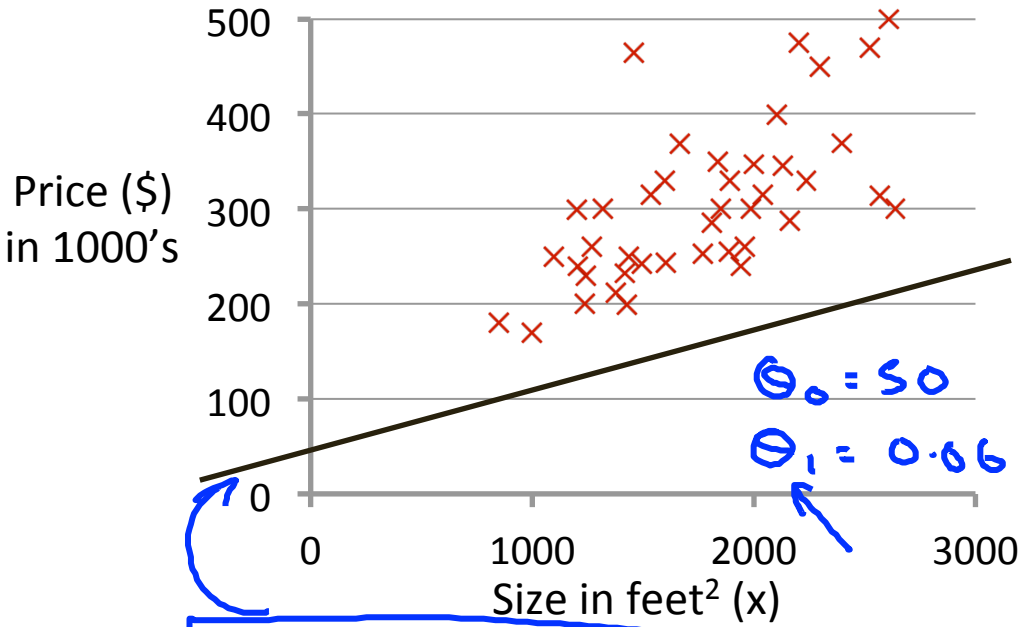
Parameters: θ_0, θ_1

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

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

$h_\theta(x)$

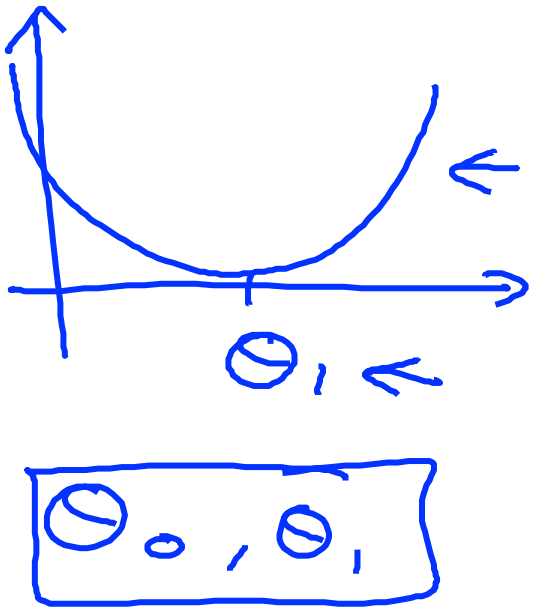
(for fixed θ_0, θ_1, this is a function of x)



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

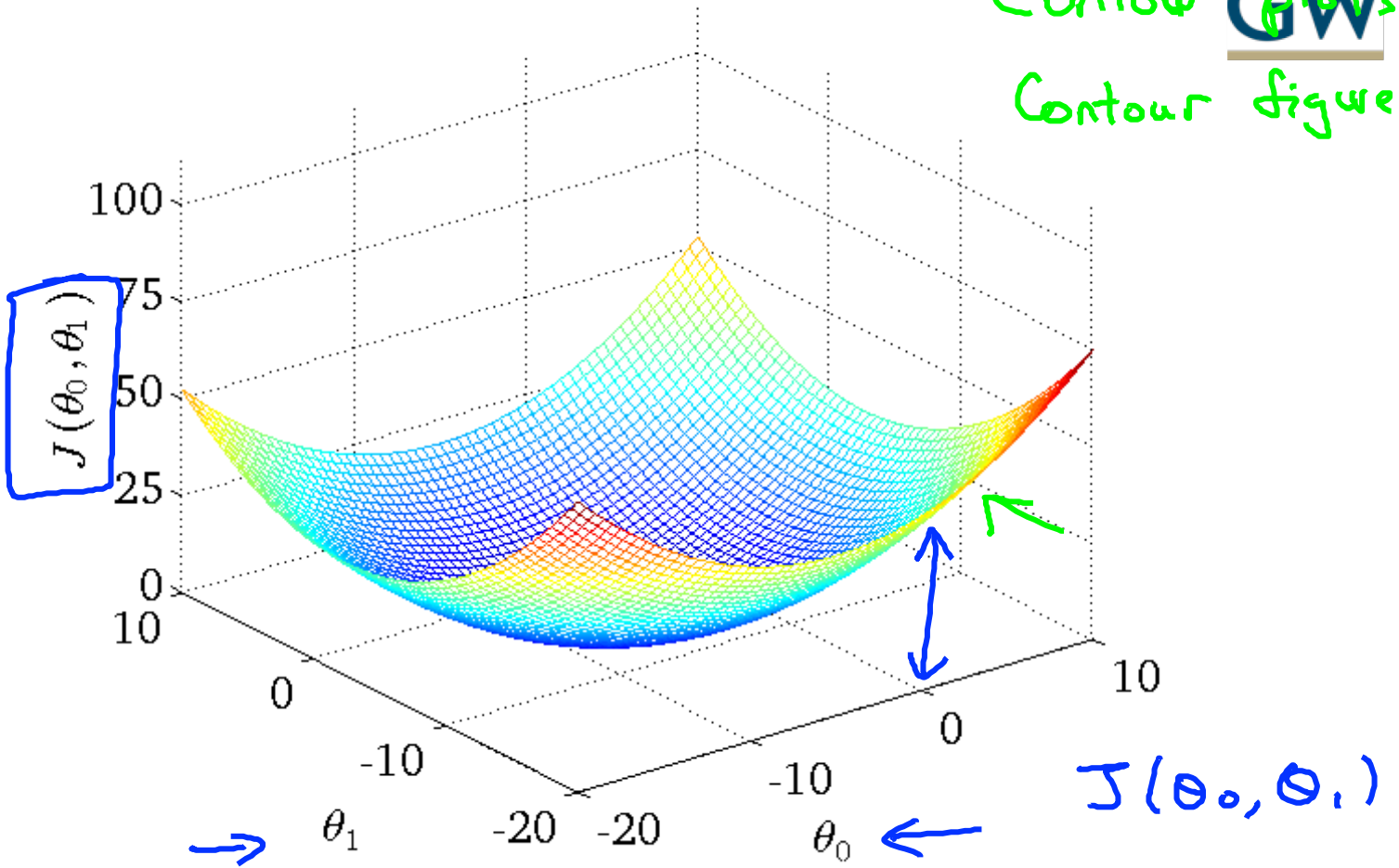
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Contour plots

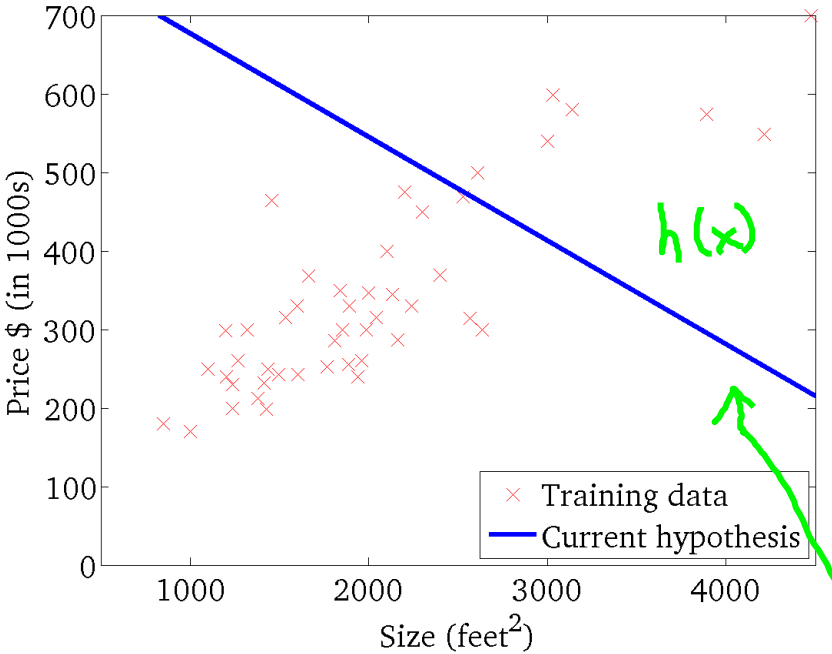
Contour figures -



$J(\theta_0, \theta_1)$

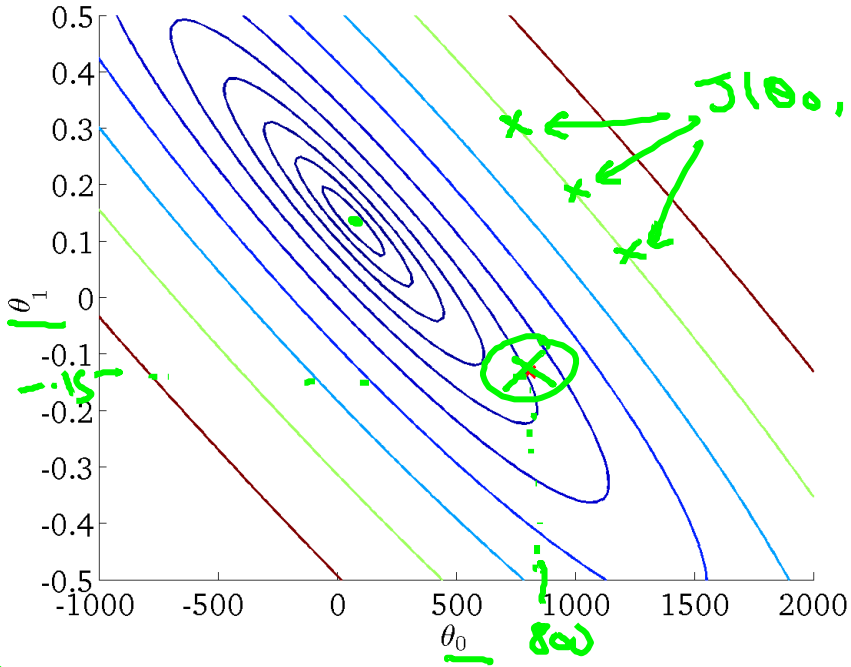
$h_\theta(x)$

(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)

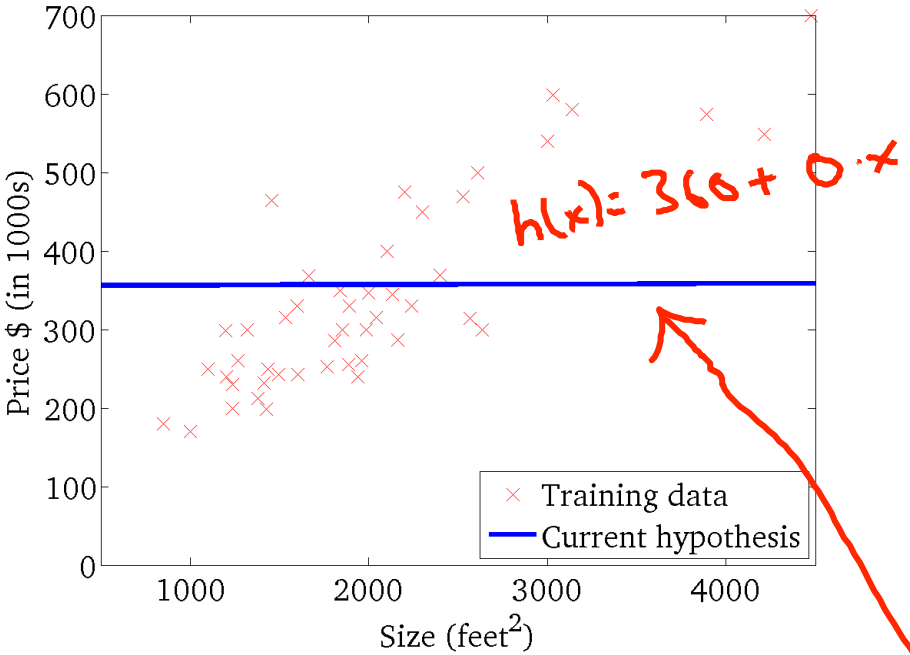


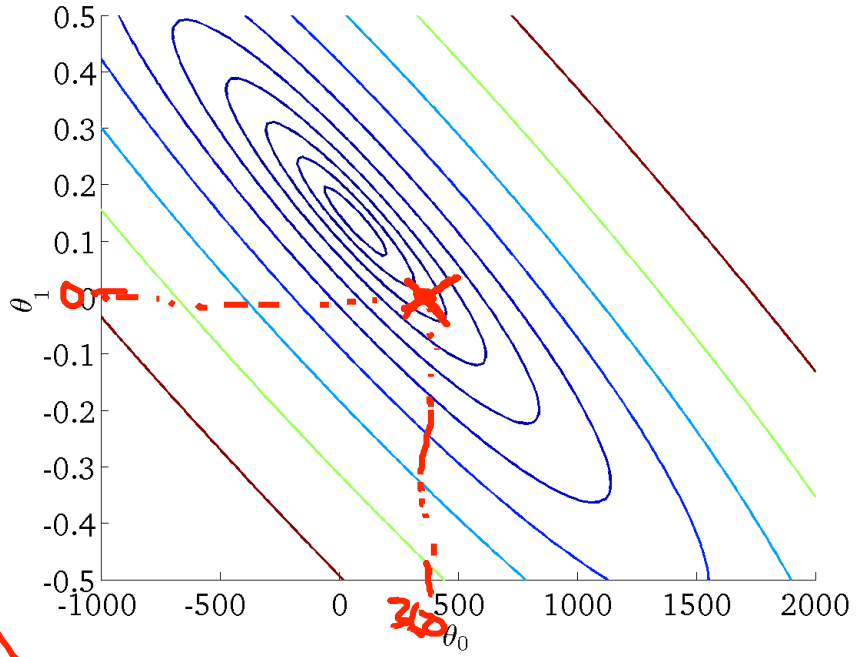
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

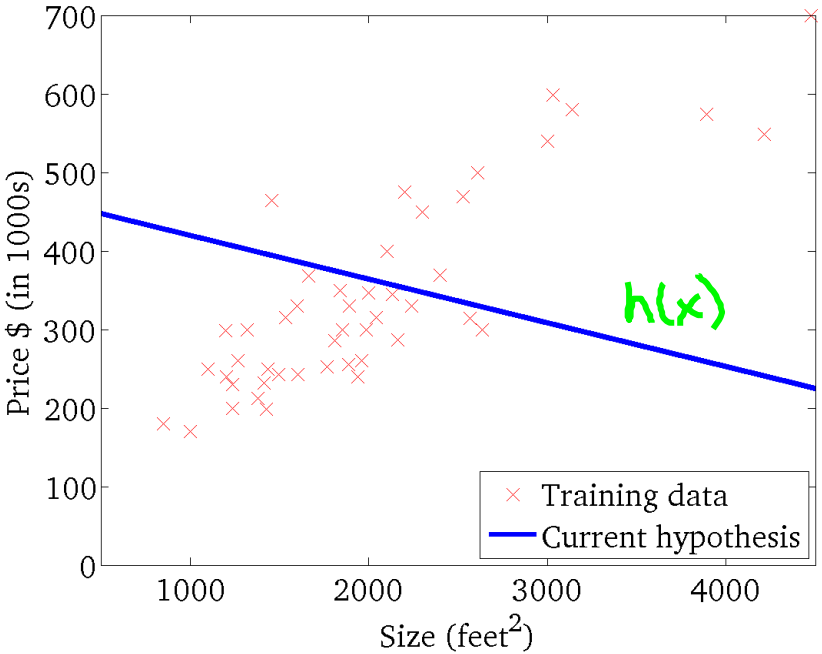
(function of the parameters θ_0, θ_1)





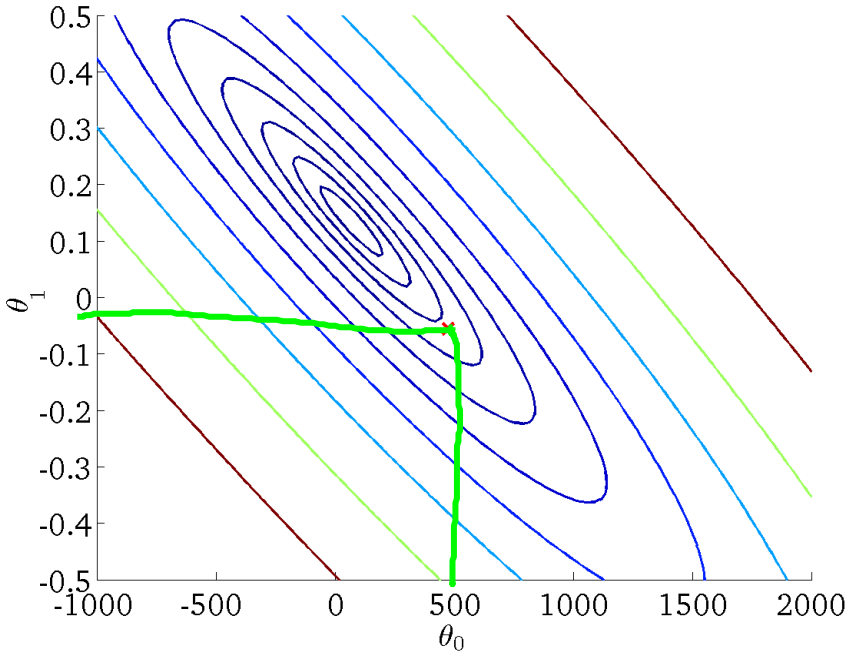
$h_\theta(x)$

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



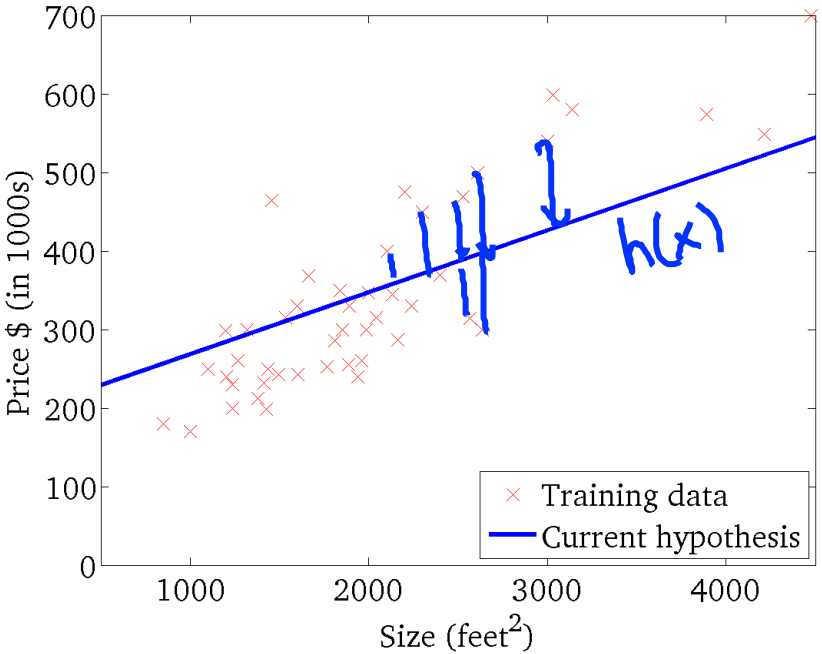
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



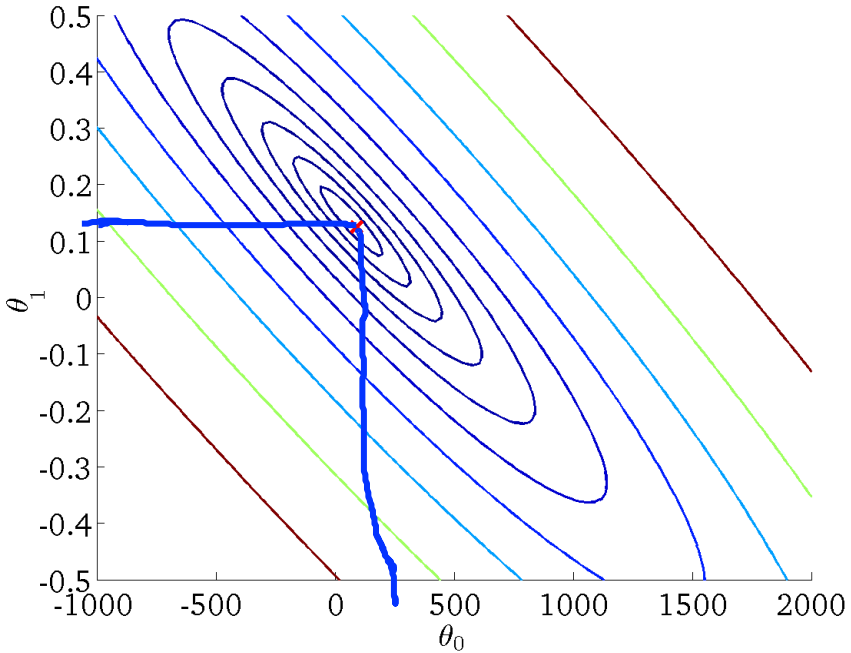
$h_\theta(x)$

(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)



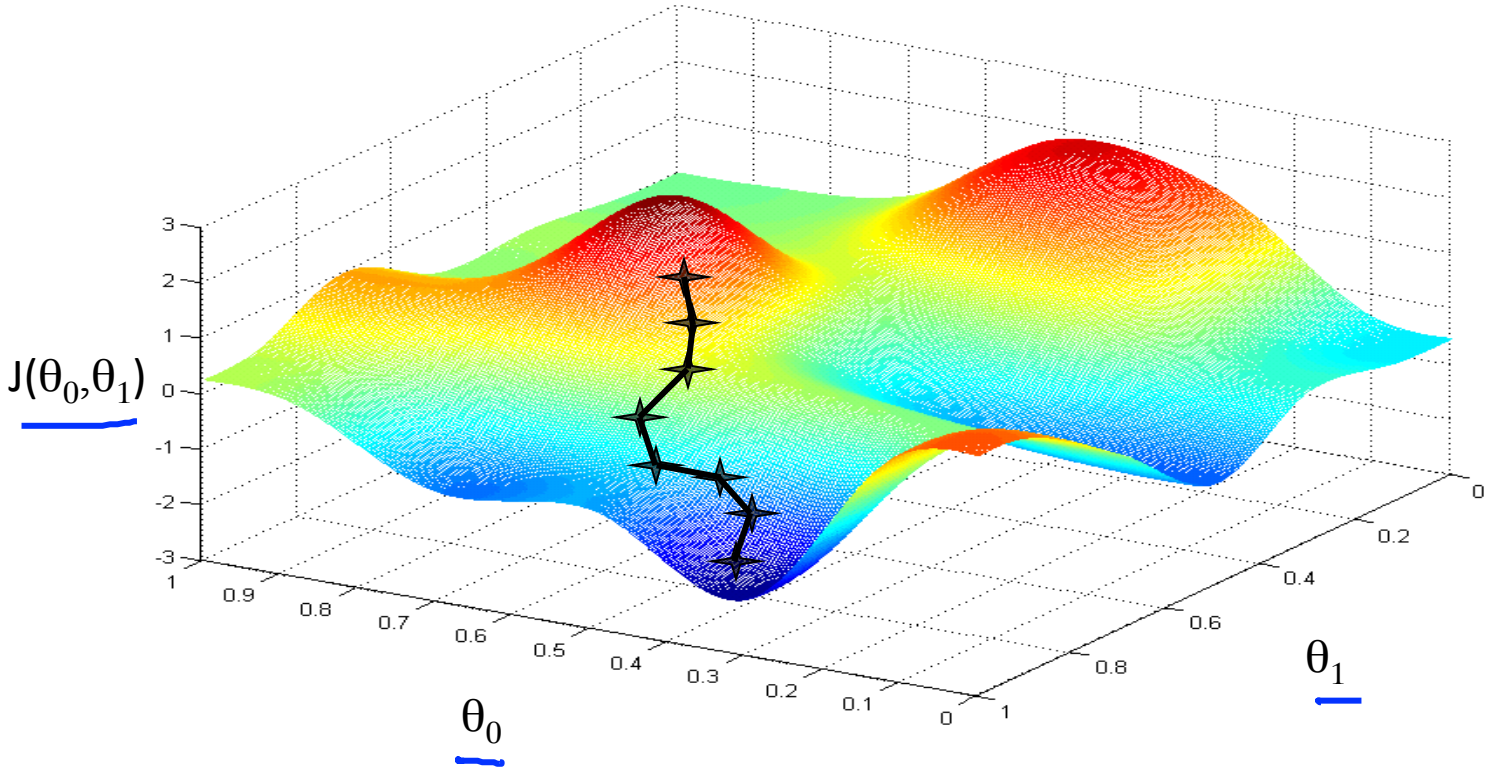
Have some function $J(\theta_0, \theta_1)$

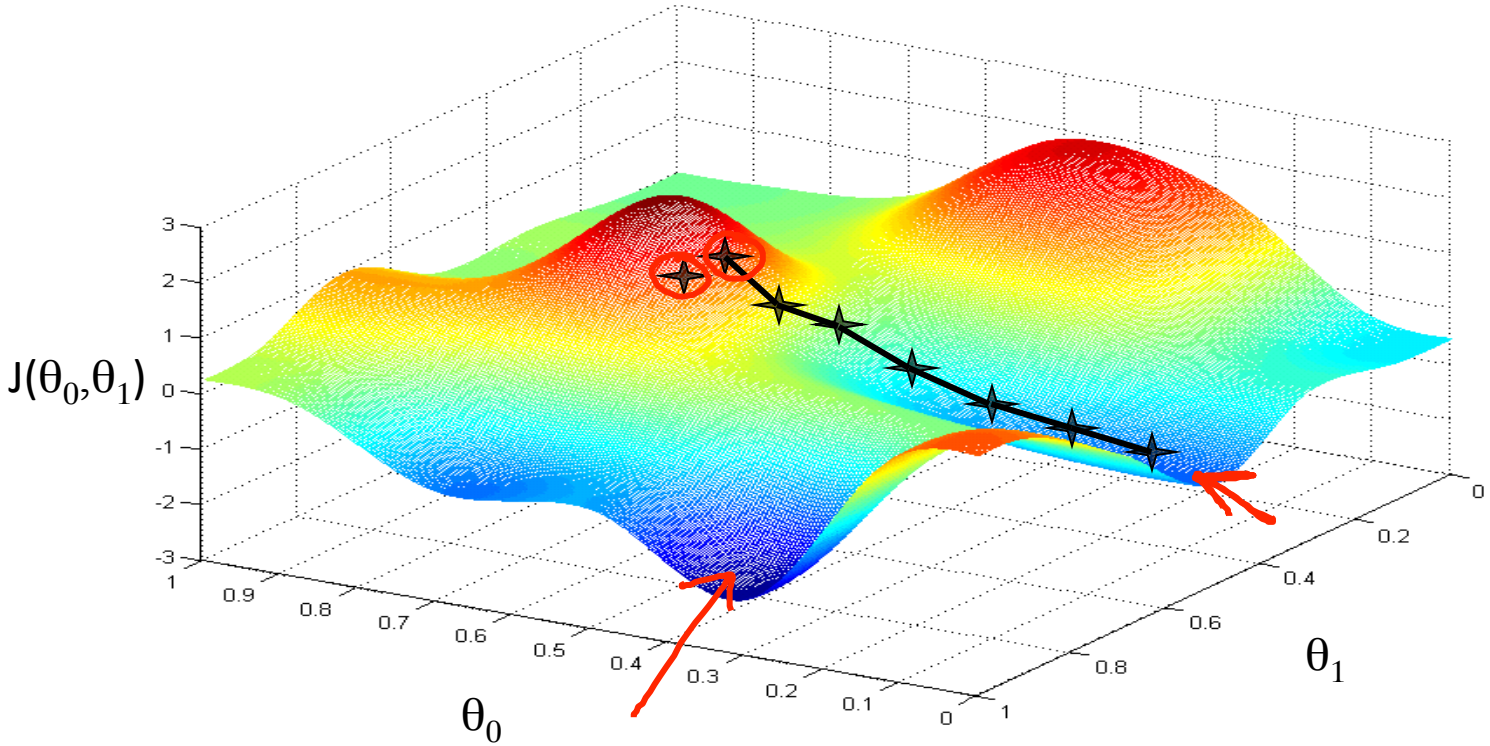
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

  until we hopefully end up at a minimum


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1

Gradient descent algorithm

repeat until convergence {

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

}

learning rate

Simultaneously update θ_0 and θ_1

Assignment

$a := b$

$a := a + 1$

Truth assertion

$a = b$

$a = a + 1$ ✗

Correct: Simultaneous update

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

Incorrect:

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

Gradient descent algorithm

repeat until convergence {

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

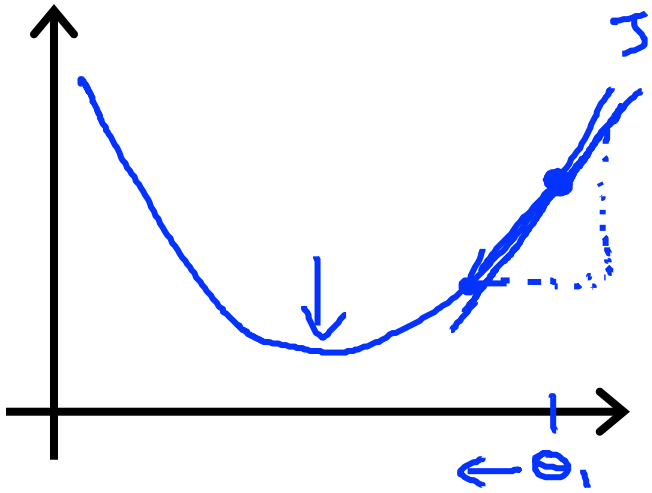
(simultaneously update $j = 0$ and $j = 1$)

}

learning rate

derivative

$\min_{\theta_1} J(\theta_1)$ $\quad \theta_1 \in \mathbb{R}$.

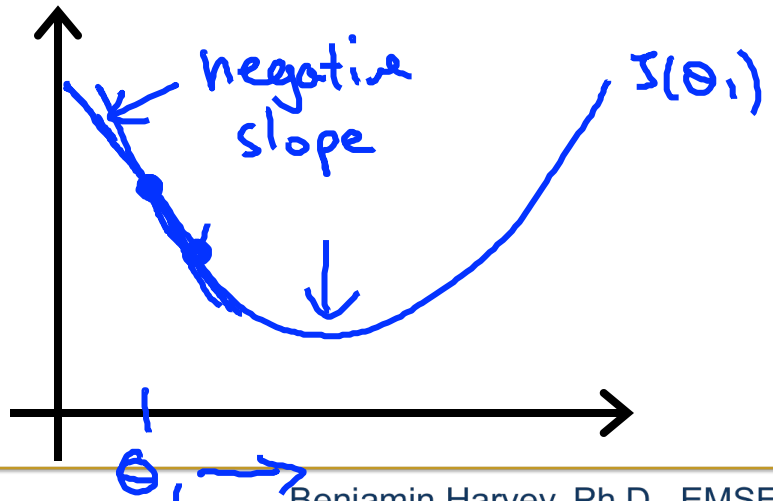


$(\theta_1 \in \mathbb{R})$

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

$\frac{d}{d\theta_1}$ ← 2

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

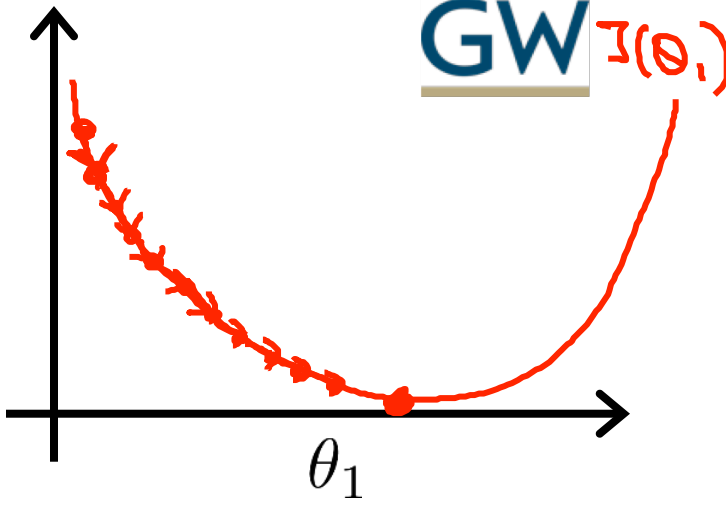


$$\underbrace{\frac{d}{d\theta_1} J(\theta_1)}_{\leq 0}$$

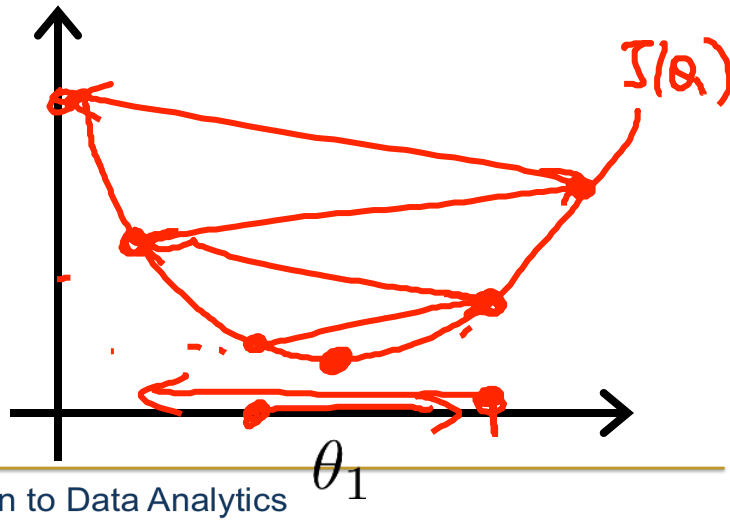
$$\theta_1 := \theta_1 - \alpha \,(\text{negative number})$$

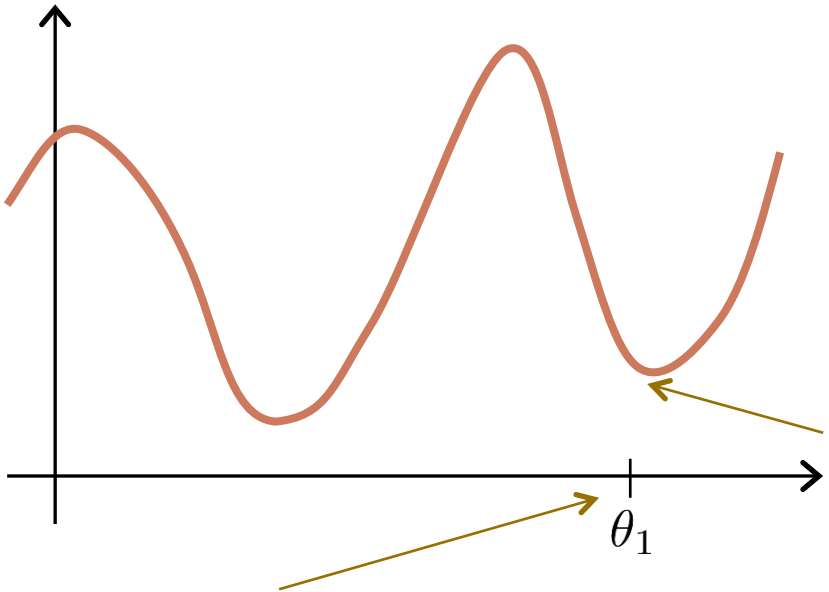
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

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.





θ_1 at local optima

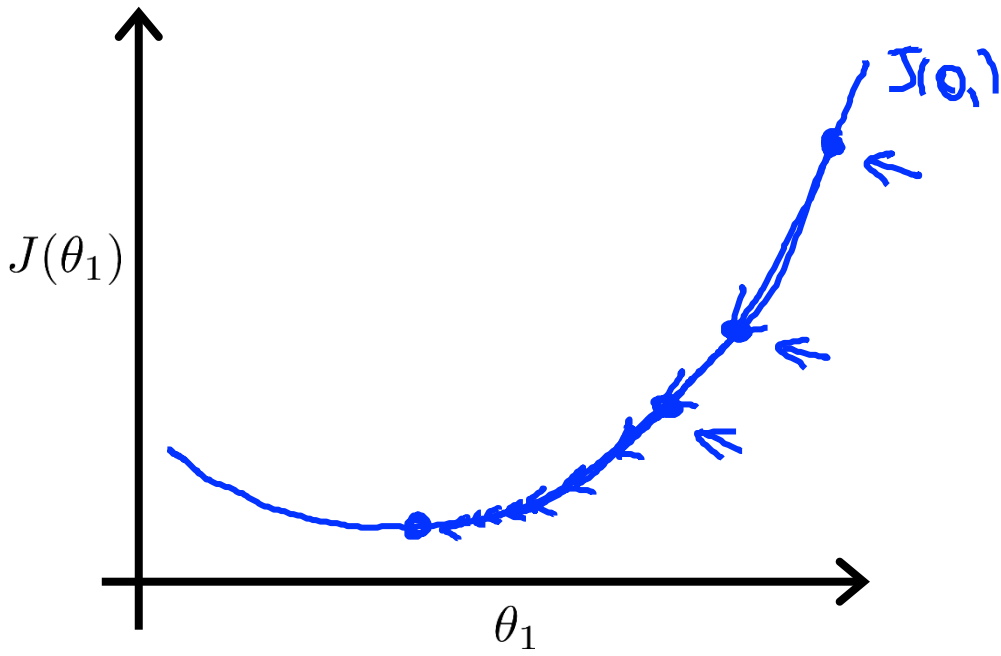
Current value of θ_1

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

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 algorithm

repeat until convergence {

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

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

}

Linear Regression Model

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

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

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

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

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

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

Gradient descent algorithm

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

repeat until convergence {

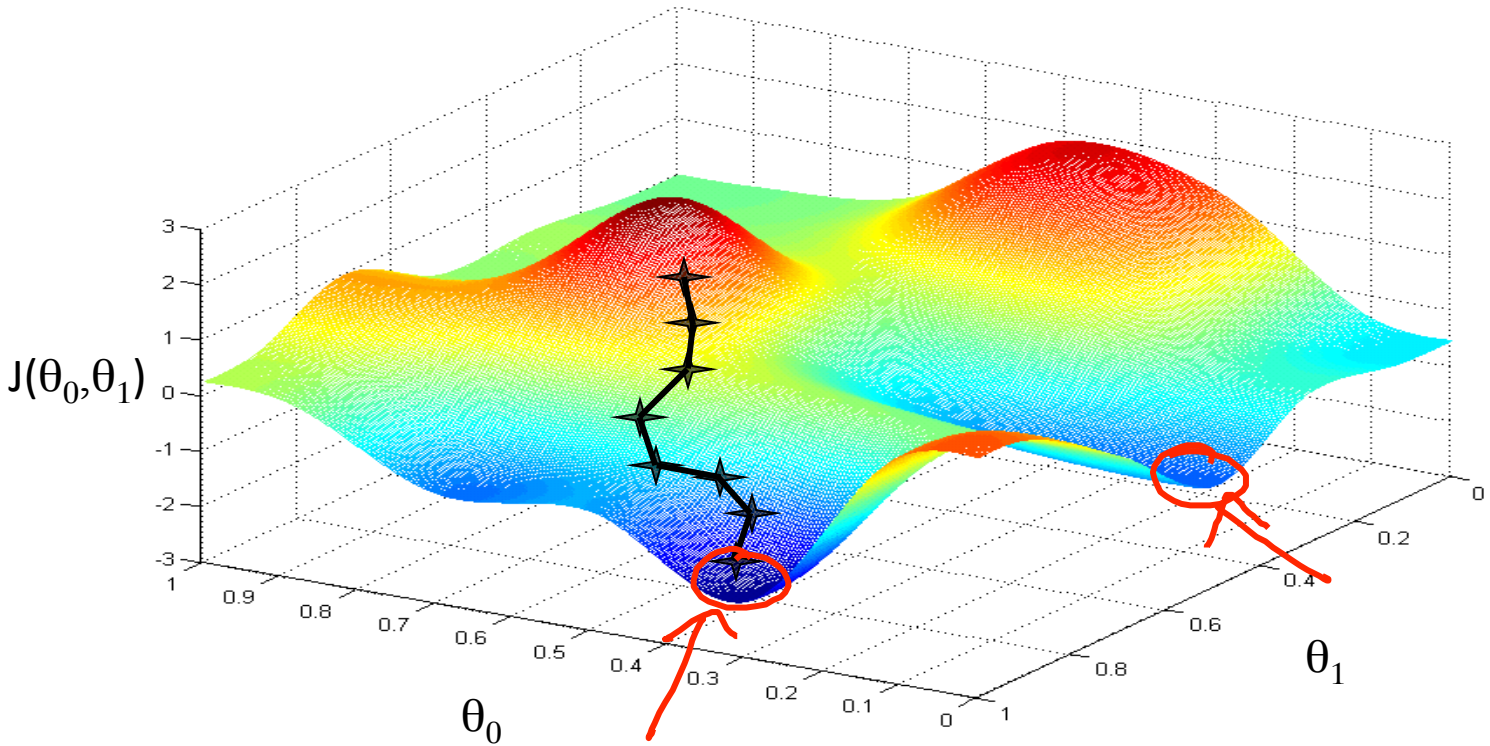
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

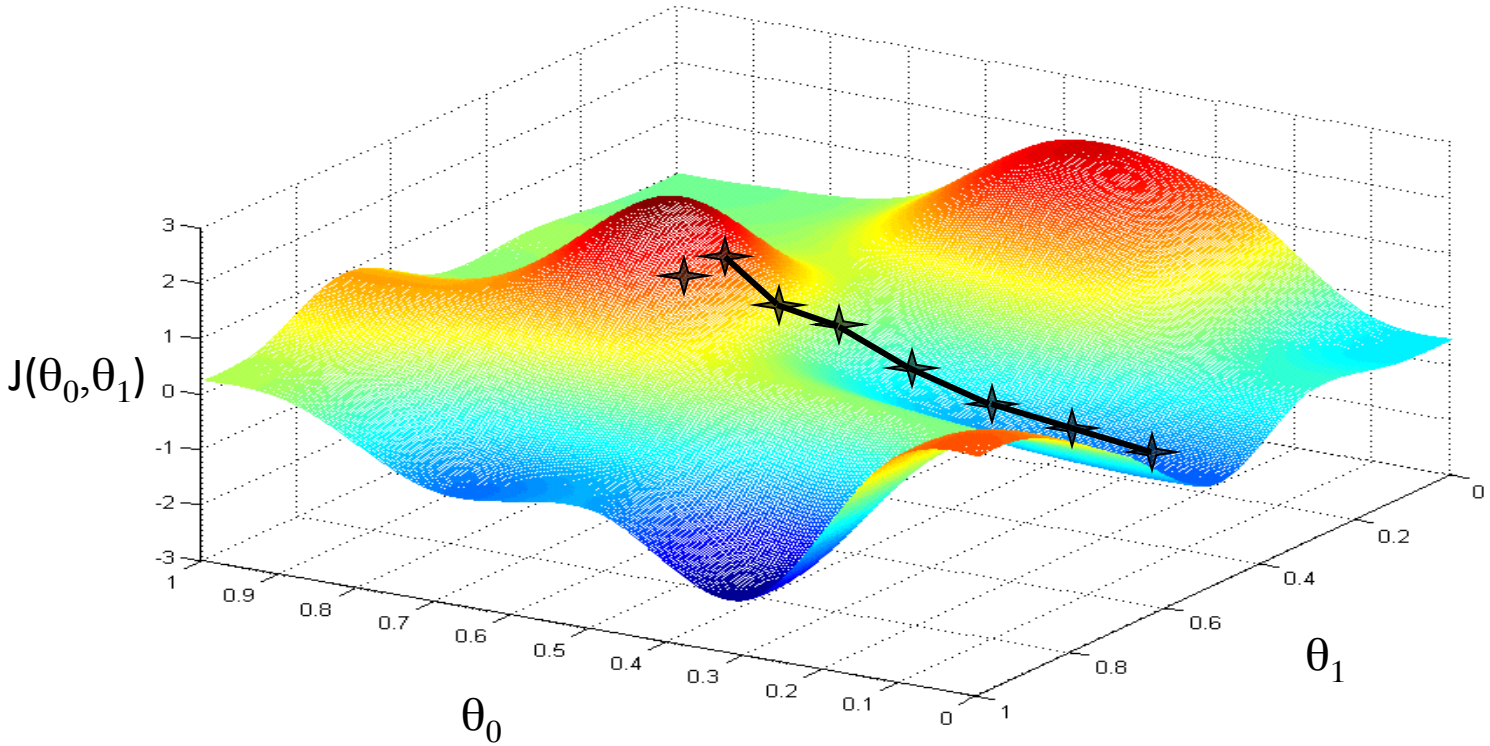
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

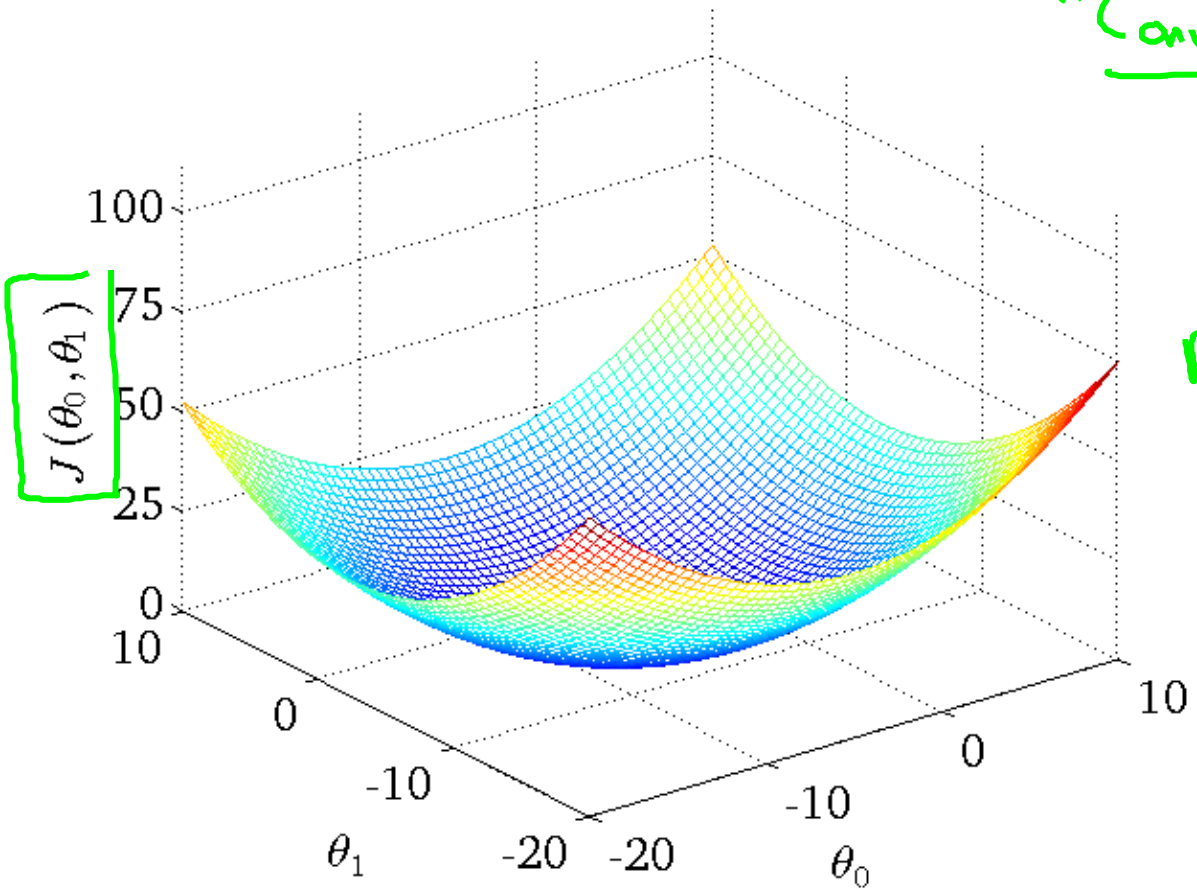
update θ_0 and θ_1 simultaneously

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


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1

"Convex Function"



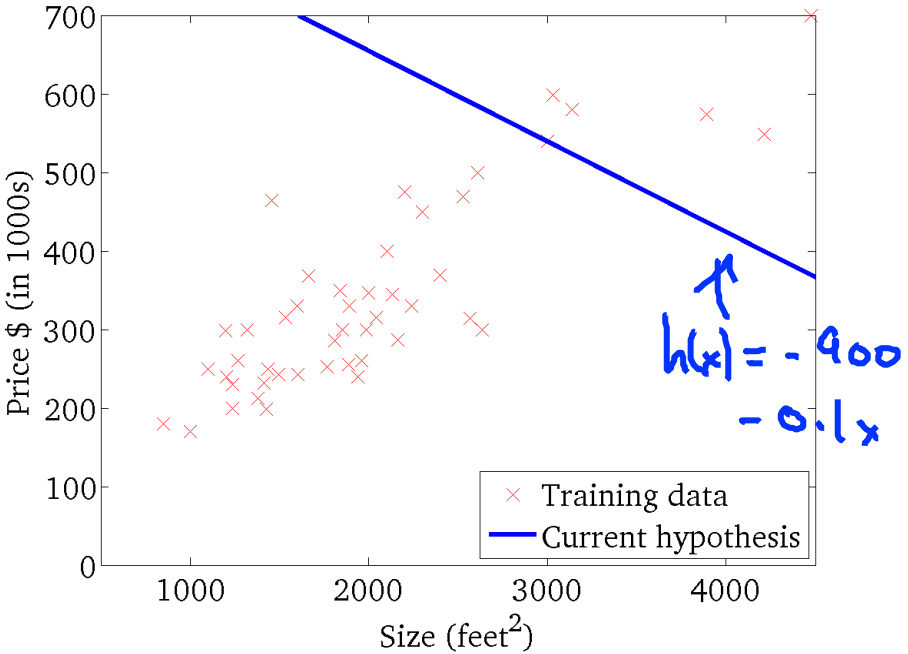
Bowl-shaped

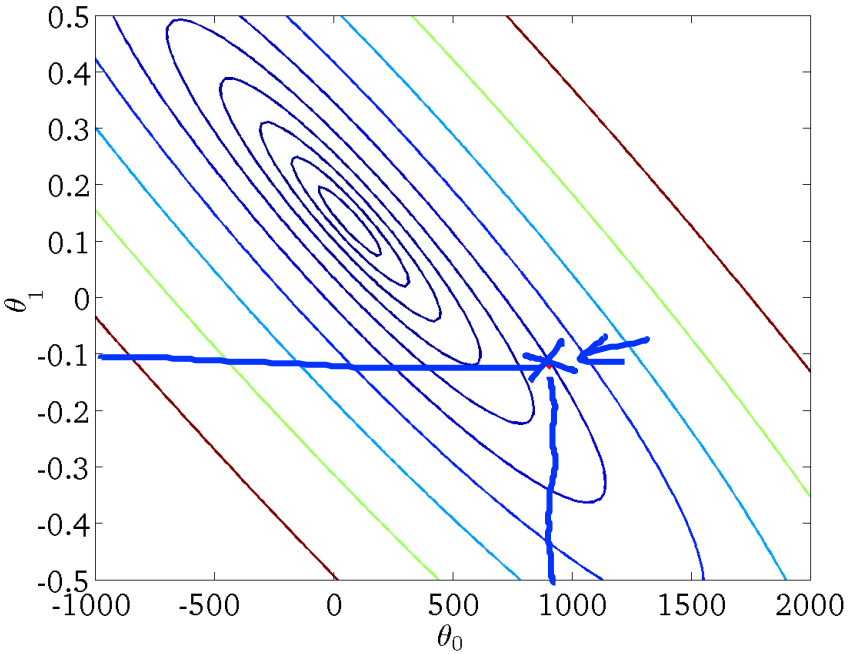
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

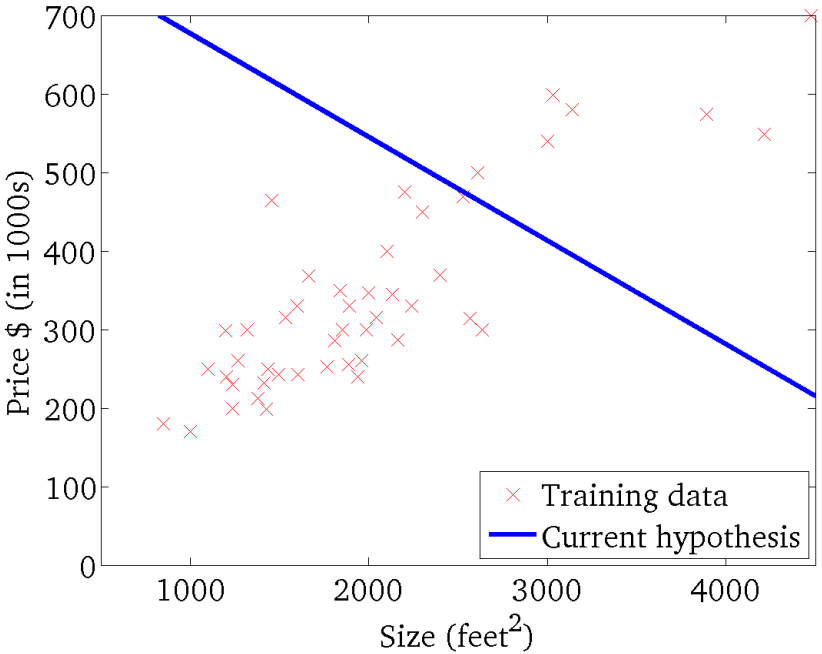
(function of the parameters θ_0, θ_1)





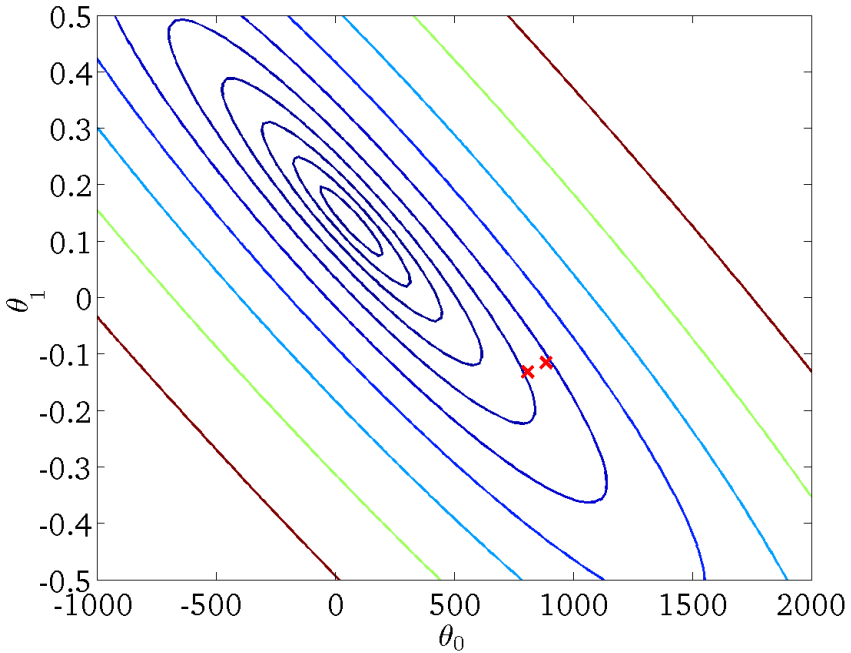
$h_\theta(x)$

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



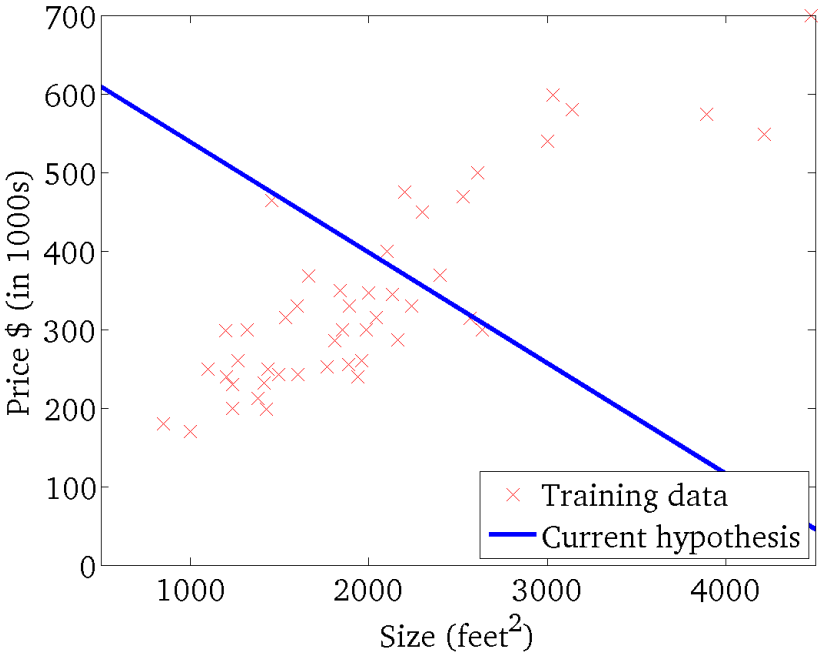
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



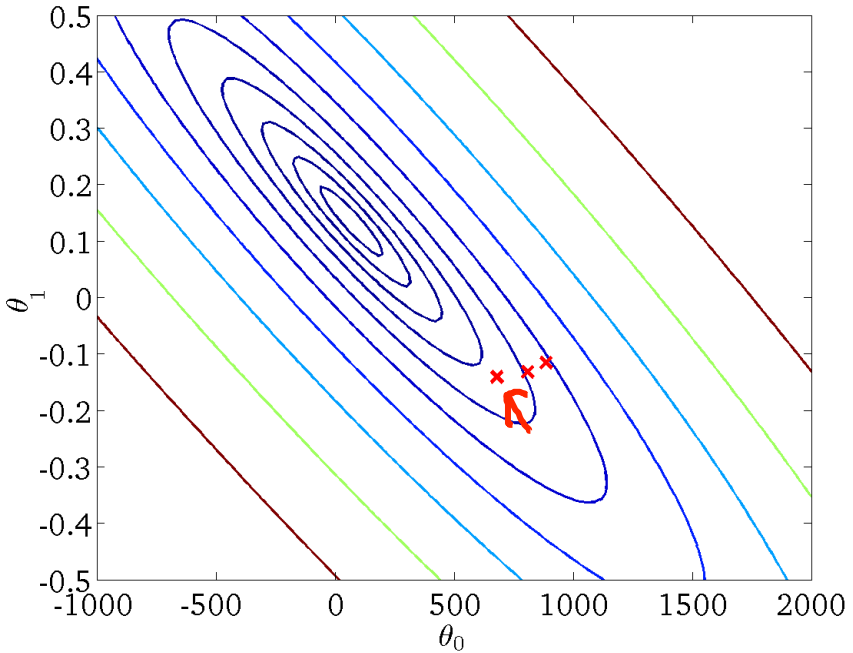
$h_\theta(x)$

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



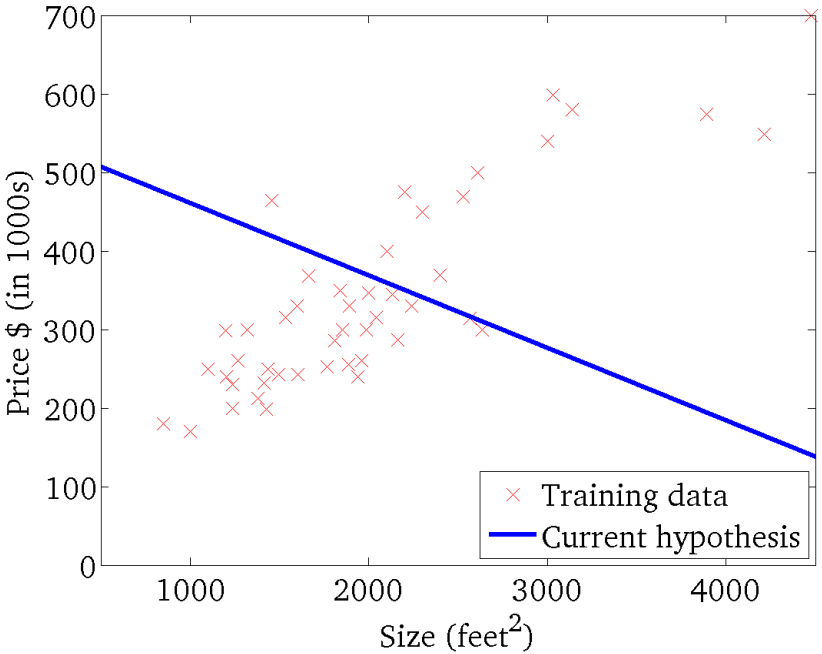
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



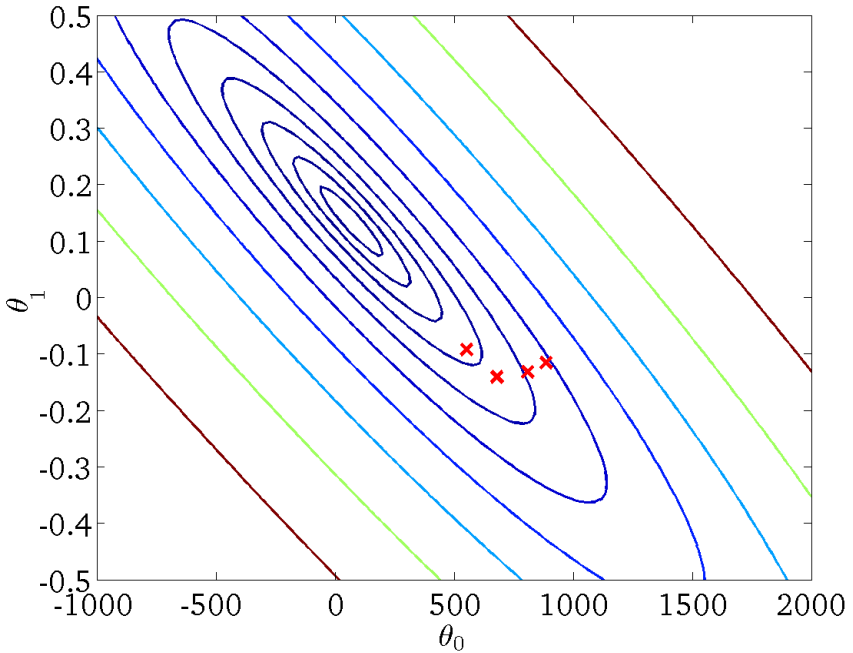
$h_\theta(x)$

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



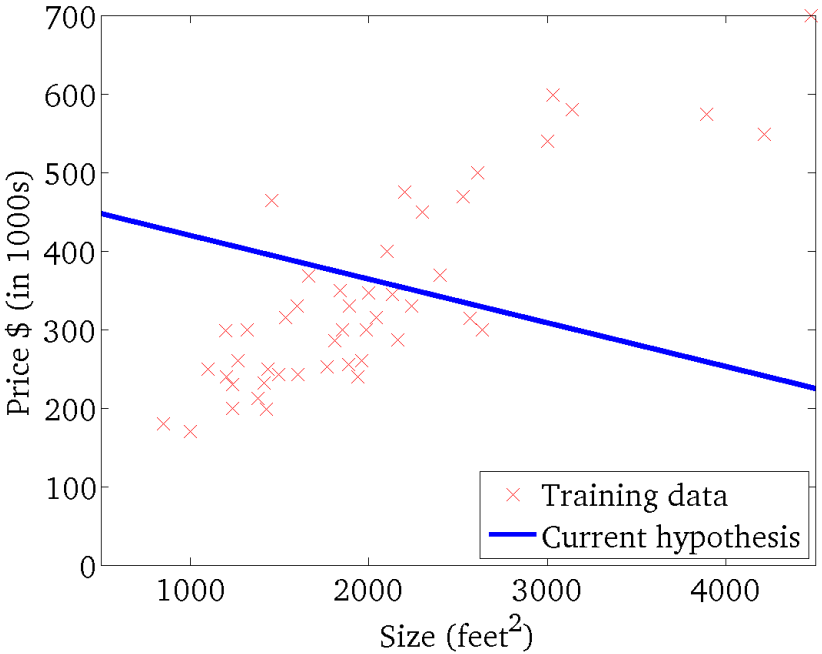
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



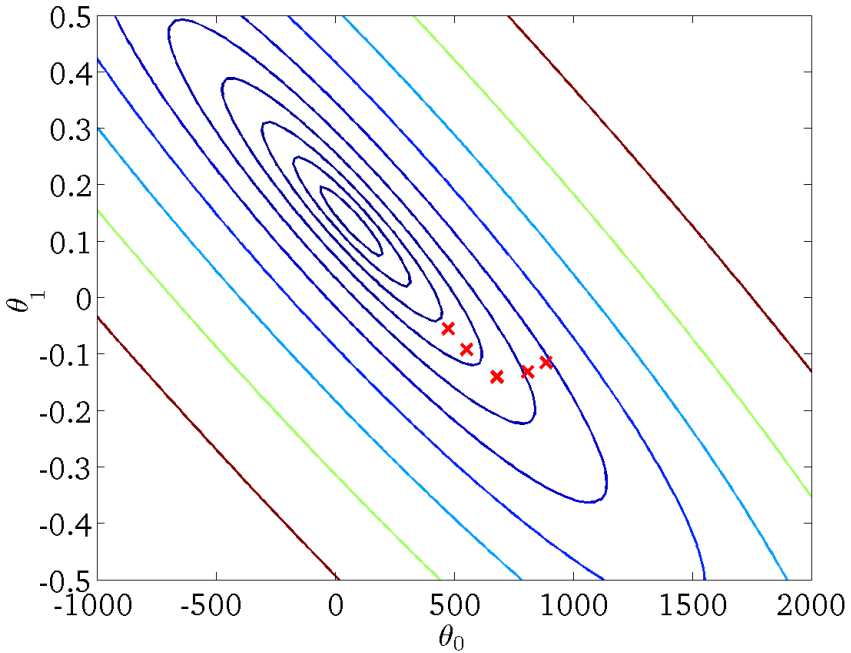
$h_\theta(x)$

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



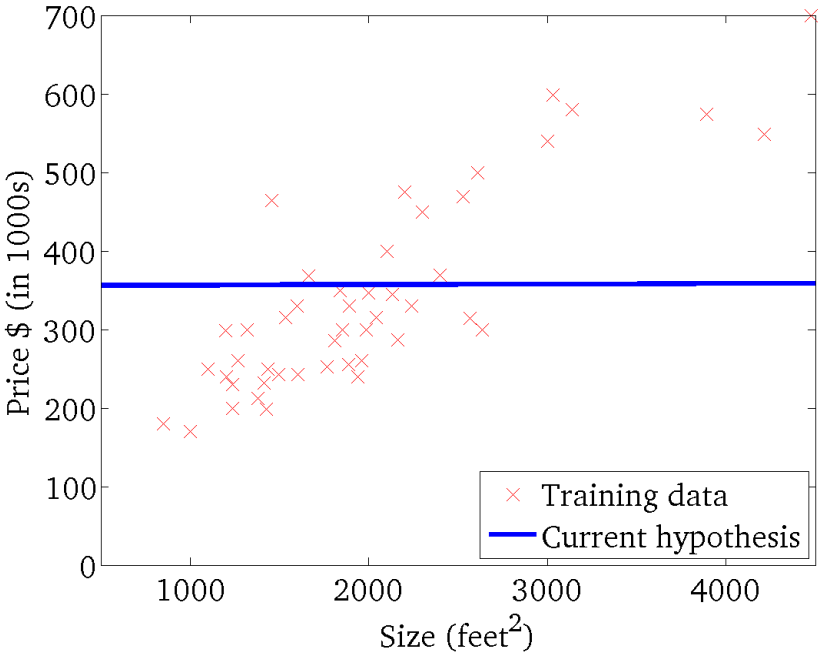
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



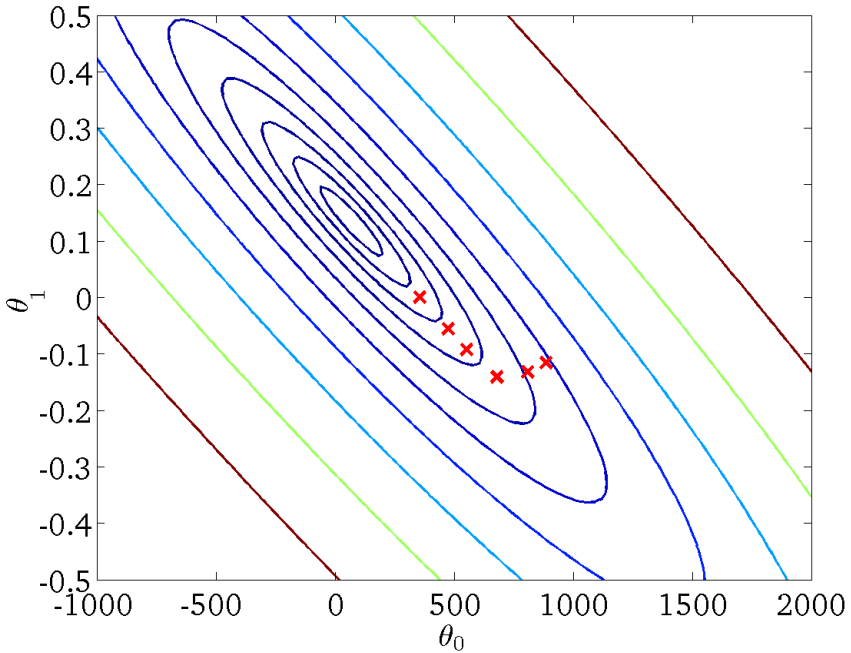
$h_\theta(x)$

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



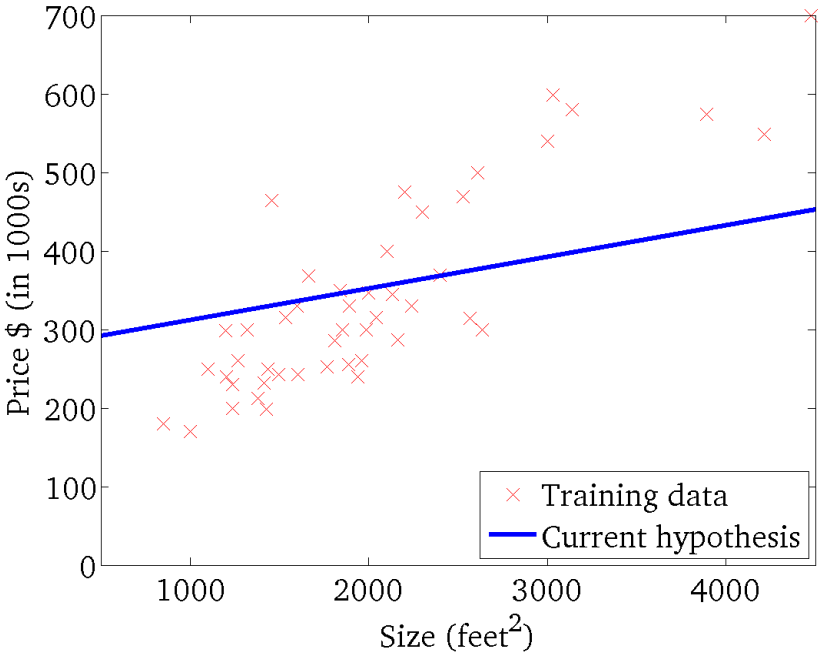
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



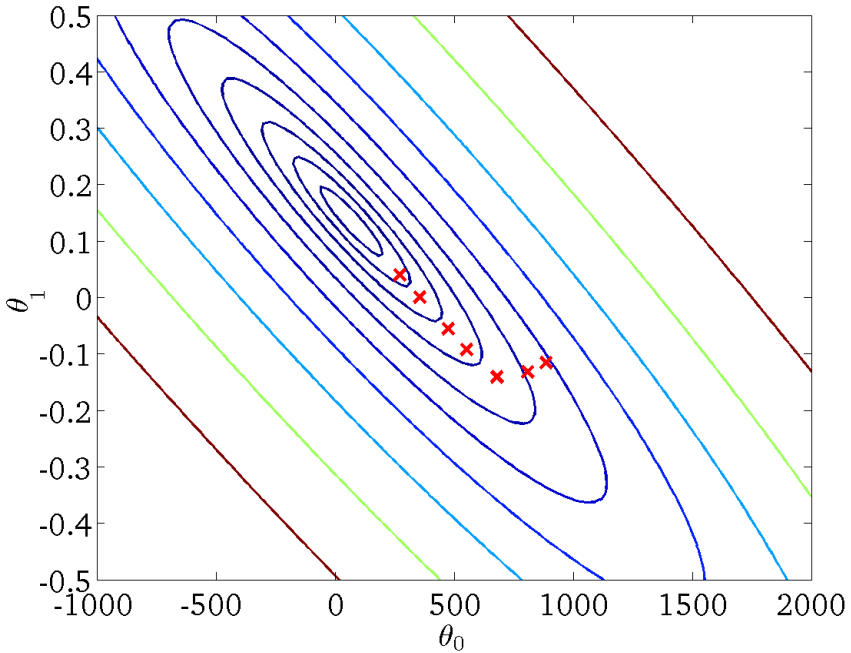
$h_\theta(x)$

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



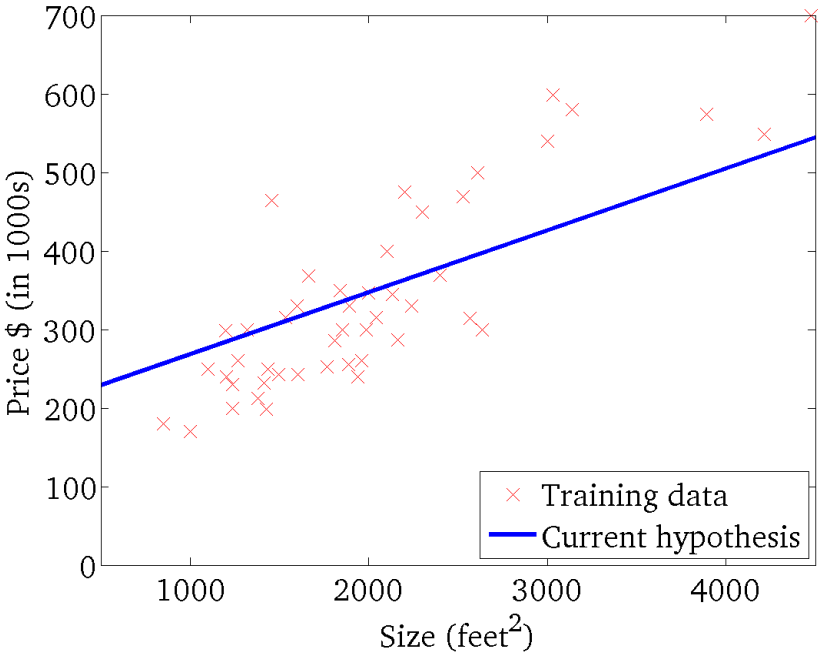
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



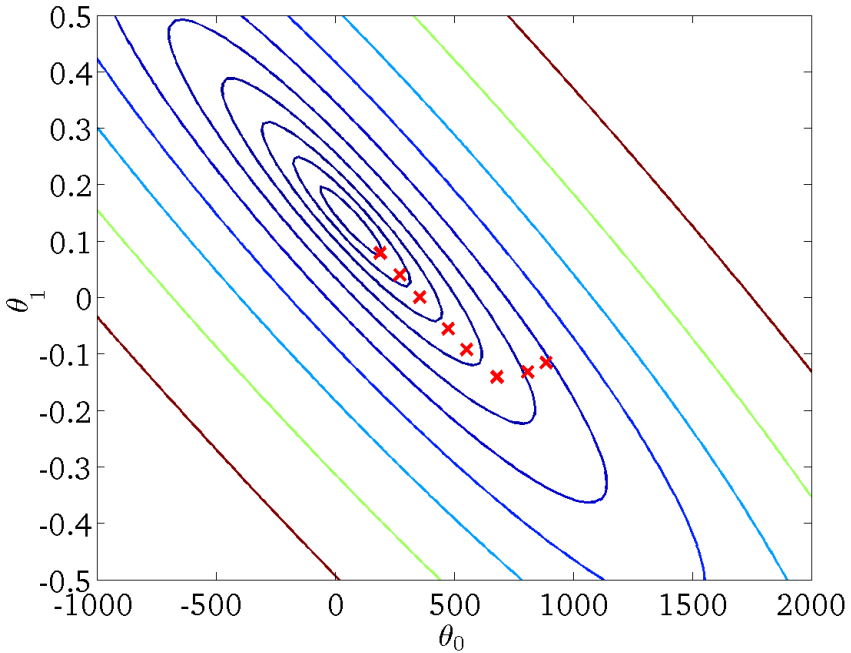
$h_\theta(x)$

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



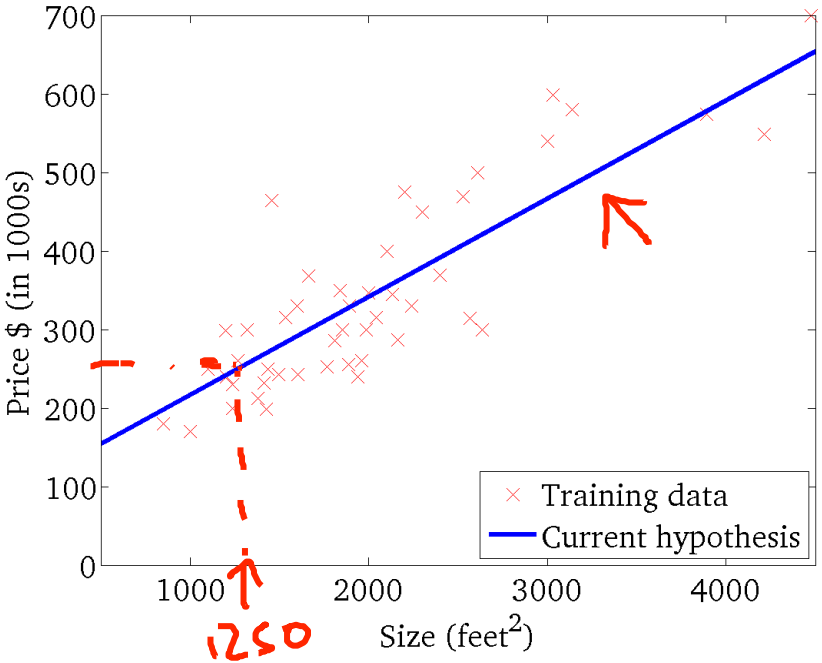
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



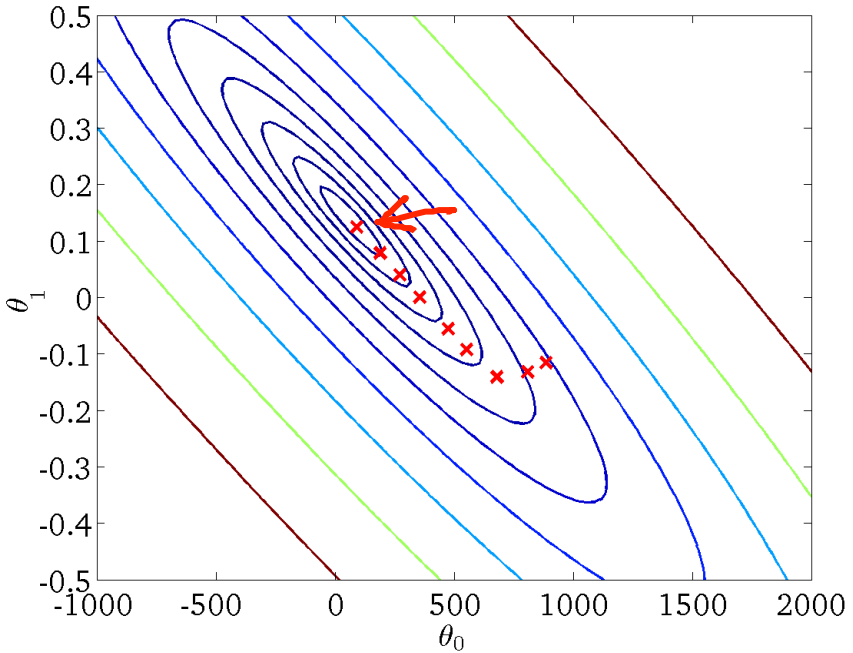
$h_\theta(x)$

(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)



“Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

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