Machine learning (Stanford) - Andrew Ng

기석, 2018. 4. 21 updataed

<Class material>

강의 : https://www.coursera.org/learn/machine-learning

Setting up programming assignment environment

- set up : http://wiki.octave.org/Octave_for_Microsoft_Windows

<Contents MAP>

2.3.3. Application

2.3.4. 주의사항

2.4. Anomaly detection

III. Special topics of Machine learning

3. Anomaly detection ← unsupervised로 분류되기도 하나 labeled data도 쓰인다.

4. Recommender systems

IV. Application details of Machine Learning

6. What ML should you apply / diagnostic

7. Machine Learning system design :

7.1. error analysis

7.2. large scale machine learning (week 10)

7.3. Generating large data (artificial data synthesis)

7.4. Online learning

8. Machine learning pipelines : Photo OCR problem (week 11)

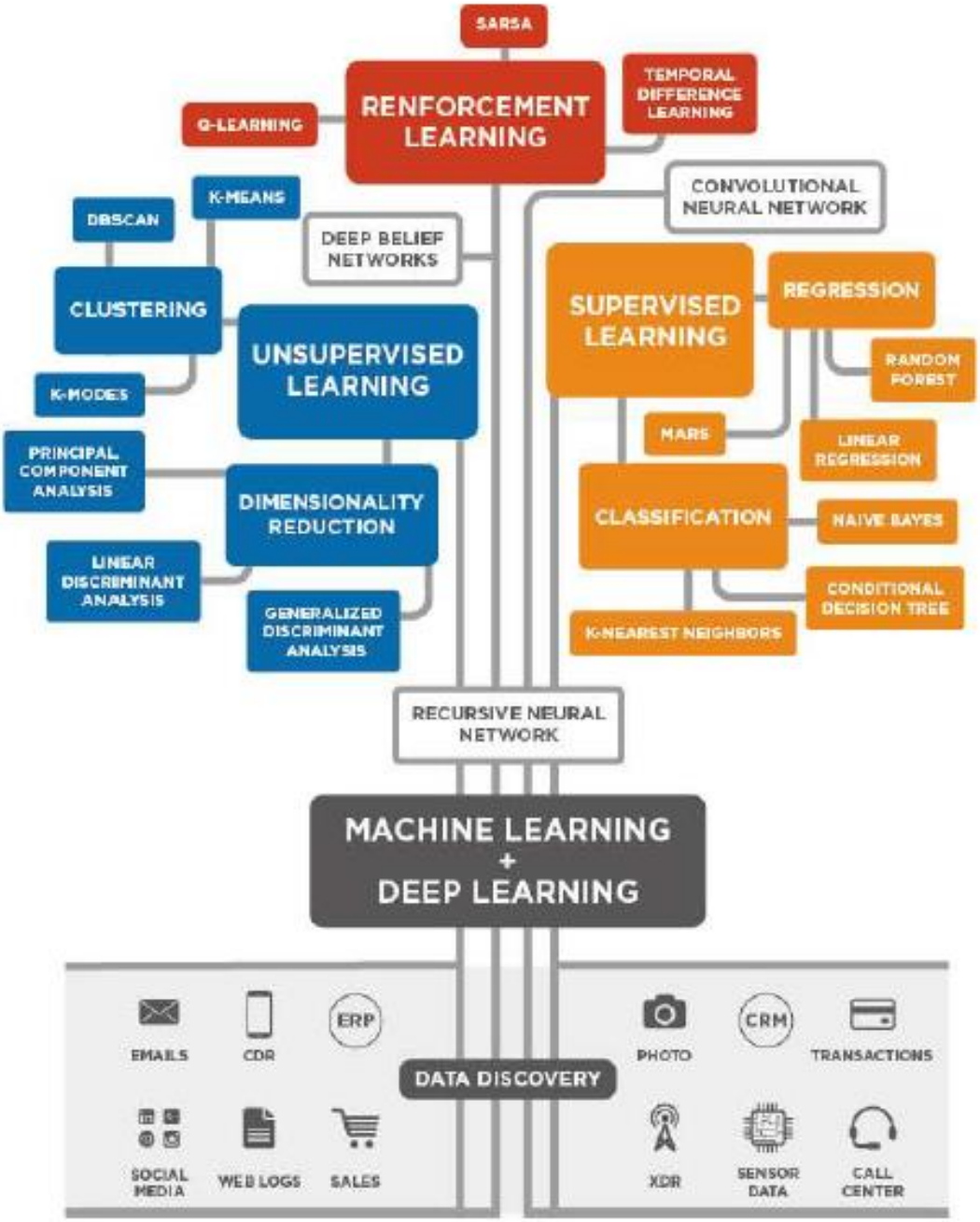
Summary: Main topics

→ Supervised Learning $(x^{(i)}, y^{(i)})$
- Linear regression, logistic regression, neural networks, SVMs

→ Unsupervised Learning $x^{(i)}$
- K-means, PCA, Anomaly detection

→ Special applications/special topics
- Recommender systems, large scale machine learning.

→ Advice on building a machine learning system
- Bias/variance, regularization; deciding what to work on next: evaluation of learning algorithms, learning curves, error analysis, ceiling analysis.

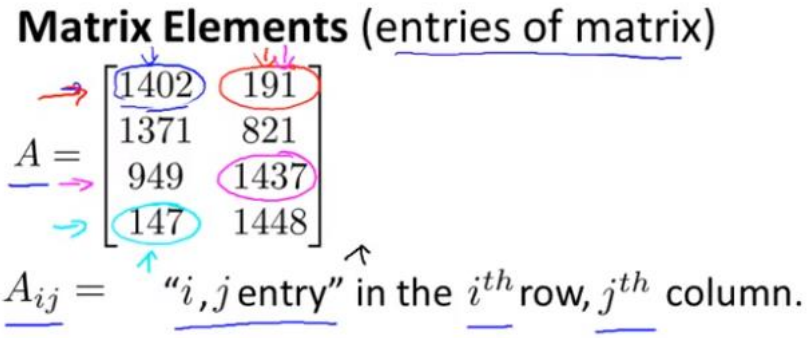

SARSA
RENFORCEMENT LEARNING
Q-LEARNING
TEMPORAL DIFFERENCE LEARNING
CONVOLUTIONAL NEURAL NETWORK
K-MEANS
DBSCAN
DEEP BELIEF NETWORKS
CLUSTERING
UNSUPERVISED LEARNING
SUPERVISED LEARNING
REGRESSION
RANDOM FOREST
K-MODES
MARS
LINEAR REGRESSION
PRINCIPAL COMPONENT ANALYSIS
DIMENSIONALITY REDUCTION
CLASSIFICATION
NAIVE BAYES
LINEAR DISCRIMINANT ANALYSIS
CONDITIONAL DECISION TREE
GENERALIZED DISCRIMINANT ANALYSIS
K-NEAREST NEIGHBORS
RECURSIVE NEURAL NETWORK
MACHINE LEARNING + DEEP LEARNING
EMAILS
CDR
ERP
PHOTO
CRM
TRANSACTIONS
DATA DISCOVERY
SOCIAL MEDIA
WEB LOGS
SALES
XDR
SENSOR DATA
CALL CENTER

I. 기본

0. 선형대수학 복습

0.1. Matrix and vector 복습

- Matrix :



Matrices are 2-dimensional arrays:

```
$$\begin{bmatrix} a & b & c \newline d & e & f \newline g & h & i \newline j & k & l\end{bmatrix}$$
```

The above matrix has four rows and three columns, so it is a 4 x 3 matrix.

- [ ] $$\begin{bmatrix} 1 & 2 \\ 4 & 0 \\ 0 & 1 \end{bmatrix}$$ is a $$3\times2$$ matrix.
- [ ] $$\begin{bmatrix} 0 & 1 & 4 & 2 \\ 3 & 4 & 0 & 9\end{bmatrix}$$ is a $$4\times2$$ matrix.
- [ ] $$\begin{bmatrix} 0 & 4 & 2 \\ 3 & 4 & 9 \\ 5 & -1 & 0 \end{bmatrix}$$ is a $$3\times3$$ matrix.
- [ ] $$\begin{bmatrix}1 & 2\end{bmatrix}$$ is a $$1\times2$$ matrix.

Let A be a matrix shown below. $$A_{32}$$ is one of the elements of this matrix.

$$A = \begin{bmatrix} 85 & 76 & 66 & 5 \\ 94 & 75 & 18 & 28 \\ 68 & 40 & 71 & 5 \end{bmatrix}$$

What is the value of $$A_{32}$$?

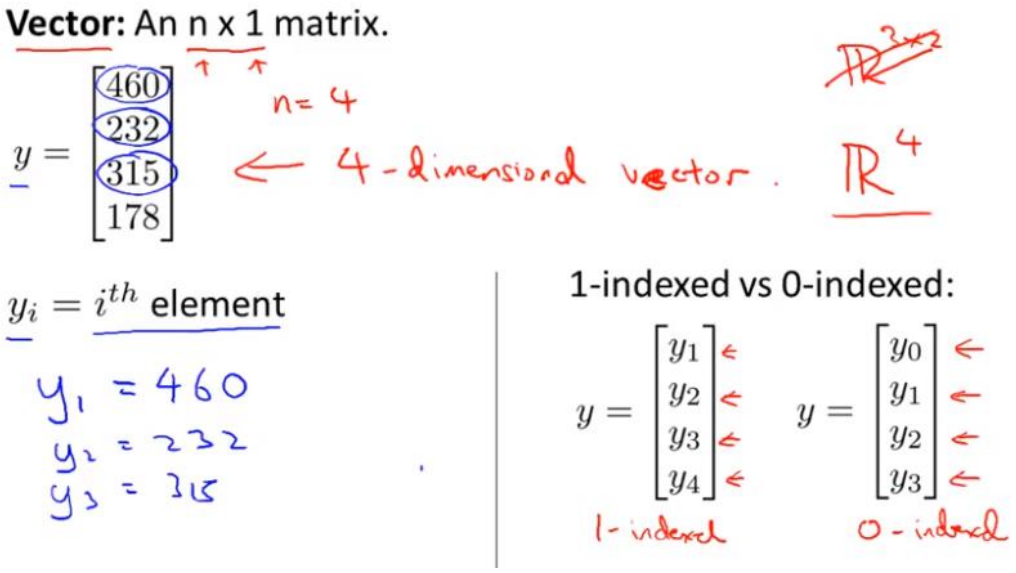
→ 40이다.

- Vector : vector is a special case of matrix

A vector is a matrix with one column and many rows:

```
$$\begin{bmatrix} w \newline x \newline y \newline z \end{bmatrix}$$
```

So vectors are a subset of matrices. The above vector is a 4 x 1 matrix.



- computer notation :

Notation and terms:

- $$A_{ij}$$ refers to the element in the ith row and jth column of matrix A.
- A vector with 'n' rows is referred to as an 'n'-dimensional vector.
- $$v_i$$ refers to the element in the ith row of the vector.
- In general, all our vectors and matrices will be 1-indexed. Note that for some programming languages, the arrays are 0-indexed.
- Matrices are usually denoted by uppercase names while vectors are lowercase.
- "Scalar" means that an object is a single value, not a vector or matrix.
- $$\mathbb{R}$$ refers to the set of scalar real numbers.
- $$\mathbb{R^n}$$ refers to the set of n-dimensional vectors of real numbers.

Run the cell below to get familiar with the commands in Octave/Matlab. Feel free to create matrices and vectors and try out different things.

```
% The ; denotes we are going back to a new row.
A = [1, 2, 3; 4, 5, 6; 7, 8, 9; 10, 11, 12]

% Initialize a vector
v = [1;2;3]

% Get the dimension of the matrix A where m = rows and n = columns
[m,n] = size(A)

% You could also store it this way
dim_A = size(A)

% Get the dimension of the vector v
dim_v = size(v)

% Now let's index into the 2nd row 3rd column of matrix A
A_23 = A(2,3)
```

Run

Reset

```
A =

    1    2    3
    4    5    6
    7    8    9
   10   11   12

v =

    1
    2
    3

m =  4
n =  3
dim_A =

    4    3

dim_v =

    3    1

A_23 =  6
```

0.2. Matrix addition & scalar multiplication

→ matrix dimension이 같아야 한다.

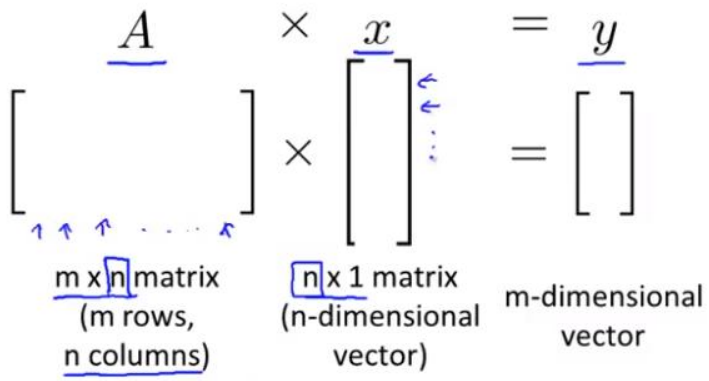
```
% Initialize matrix A and B
A = [1, 2, 4; 5, 3, 2]
B = [1, 3, 4; 1, 1, 1]

% Initialize constant s
s = 2

% See how element-wise addition works
add_AB = A + B

% See how element-wise subtraction works
sub_AB = A - B

% See how scalar multiplication works
mult_As = A * s

% Divide A by s
div_As = A / s

% What happens if we have a Matrix + scalar?
add_As = A + s
```

Run

Reset

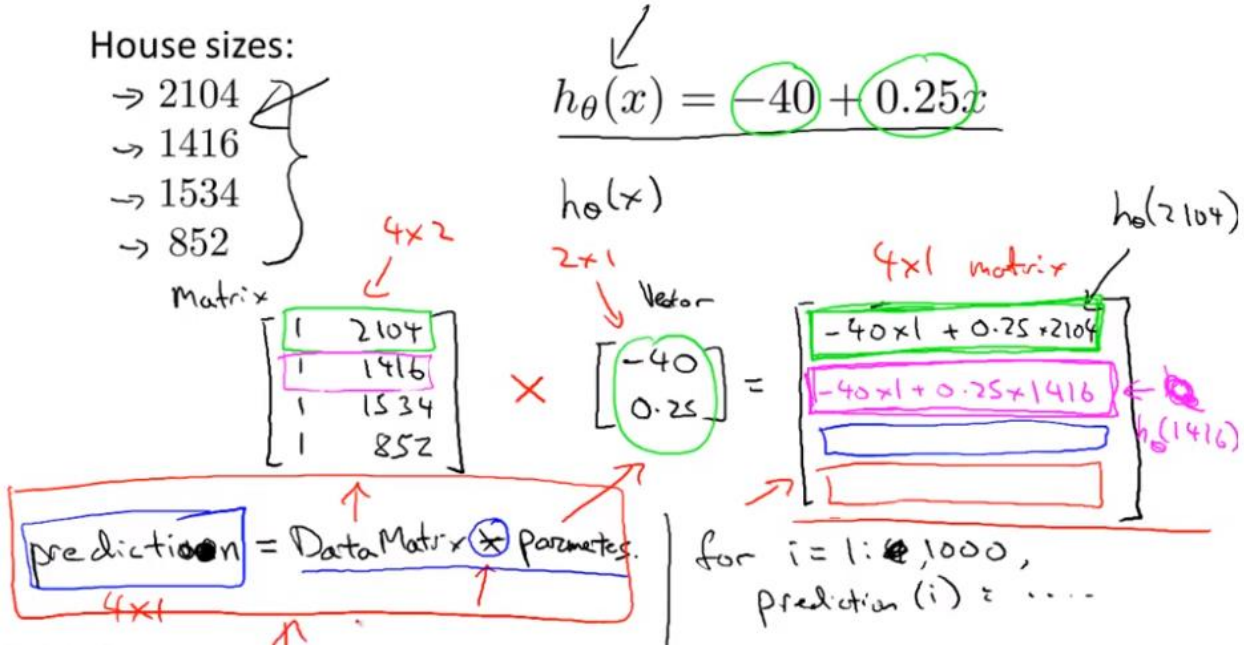
```
A =

   1   2   4
   5   3   2

B =

   1   3   4
   1   1   1

s =  2
add_AB =

   2   5   8
   6   4   3

sub_AB =

   0  -1   0
   4   2   1

mult_As =

    2    4    8
   10    6    4

div_As =

   0.50000   1.00000   2.00000
   2.50000   1.50000   1.00000

add_As =

   3   4   6
   7   5   4
```

0.3. Matrix vector multiplication



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

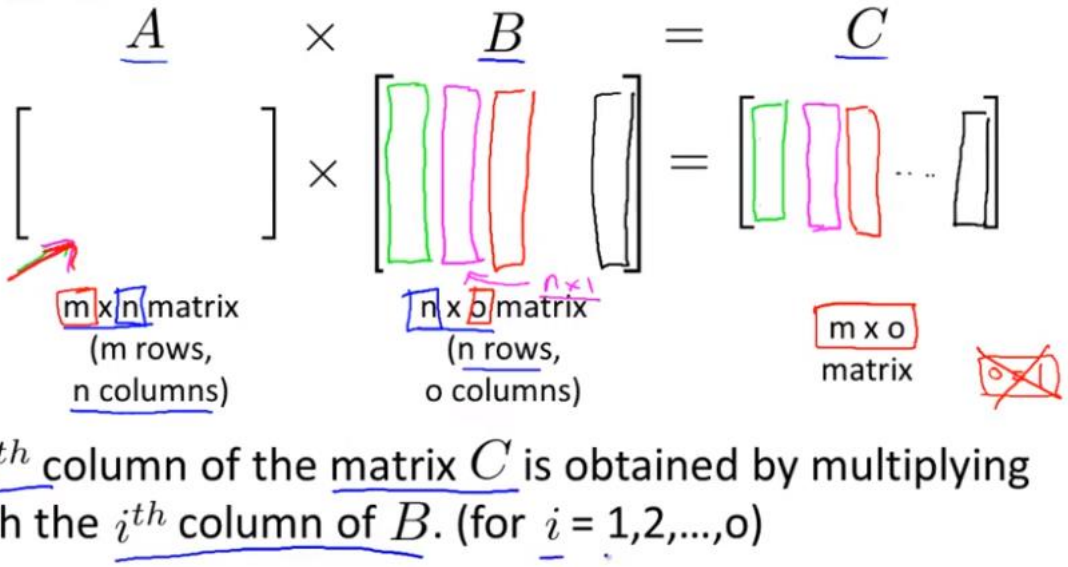
- unit 2에서 한 거에 적용



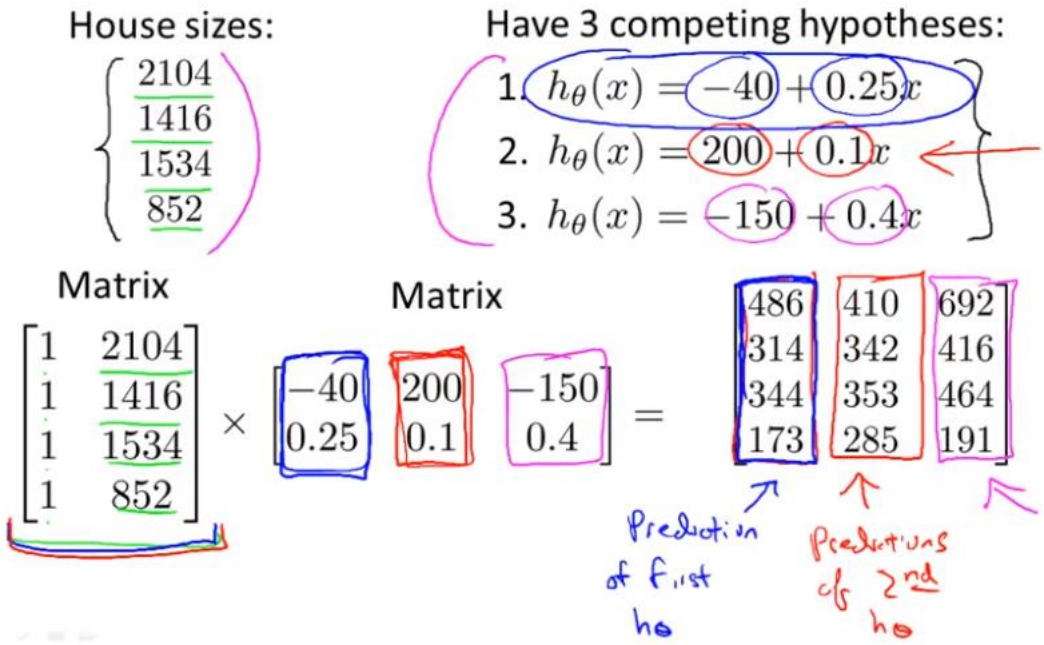
→ 1000개의 데이터가 있으면 for loop(오른쪽)보다는 matrix로 한 줄(왼쪽)로 처리하는 것이 좋다. 이 경우에는 h를 matrix-vector multiplication으로 처리했다.

0.4. Matrix-matrix multiplication

- 이것은 matrix-vector multiplication의 확장판이다.



- 적용 : 여러 hypothesis를 동시에 할 수 있다.

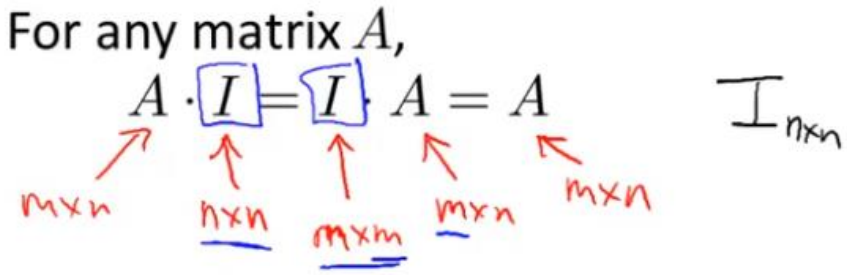


- 특징(Properties)

(1) not commutative : A * B != B * A ➔ 교환법칙 X

(2) Associative : A * (B * C) = (A * B) * C ➔ 결합법칙

(3) Identity matrix




```
% Initialize random matrices A and B
A = [1,2;4,5]
B = [1,1;0,2]

% Initialize a 2 by 2 identity matrix
I = eye(2)

% The above notation is the same as I = [1,0;0,1]

% What happens when we multiply I*A ?
IA = I*A

% How about A*I ?
AI = A*I

% Compute A*B
AB = A*B

% Is it equal to B*A?
BA = B*A

% Note that IA = AI but AB != BA
```

(4) Inverse matrix

Not all numbers have an inverse.

Matrix inverse:

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

$$A(A^{-1}) = A^{-1}A = I.$$

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

- 코딩

pinv(A)로 하면 바로 된다.

$$A^{-1}$$도 된다.

(5) Transpose

Matrix Transpose

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

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

Then B is an n x m matrix, and

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

$B_{12} = A_{21} = 2$

$B_{32} = 9$ A_{23}

What is $$\begin{bmatrix}0 & 3 \\ 1 & 4 \end{bmatrix}^T$$?

→ ^T 이거로 하네!!

```
$$A = \begin{bmatrix} a & b \newline c & d \newline e & f \end{bmatrix}$$
```

```
$$A^T = \begin{bmatrix} a & c & e \newline b & d & f \newline \end{bmatrix}$$
```

1. Octave Coding

- Prototyping language이다!!!

1.1. Basic operations

1.1.1. 사칙연산

5+6

3-2

1/2

2^6 ← python이랑 다른 점

1 == 2

1 ~= 2 ← python이랑 다른 점 !=

&& ← and

₩₩ ← or

xor(1, 0) ← or

PS1('>> ');

The default value of the primary prompt string is "octave:₩#> ". To change it, use a command like
PS1 ("₩₩u@₩₩H> ")

1.1.2. assigning

a = 3; ← semicolon은 suppress output

>> c = (3>=1);

>> c

c = 1 ← 신기하다. True라고 나오는 것임.

1.1.3. Displaying

(1) >> a

해도 되고

(2) >> disp (a) ; ← display해도 되구

(3) C 스타일

>> disp (sprintf('2 decimals : %0.2f', a)) ← string print, 0.2f는 무슨 의미인지 알겠지 ㅋㅋㅋ python이랑 똑같잖아.

>> disp (sprint('6 decimals : %0.6f', a))

1.1.4. Matrix

(1) Matrix

>> A = [1 2; 3 4; 5 6]

A =

1 2

3 4

5 6

(1-1) indexing

>> A (3,2)

>> A(3,2)

ans = 6

- 2열을 다 불러오려면 → : 는 전부를 의미한다.

>> A(: , 2)

ans =

2
4
6

- 2행을 다 불러오려면

>> A(2,:)

ans =

3 4

- 1행이랑 3행의 애들을 전부 가져와라

>> A ([1,3] , :)

ans =

1 2
5 6

- 만약에 행렬의 모든 원소를 열벡터로 만들고 싶다.

A (:)

ans =

1
3
5
200
10
11
12
201
202
101

(1-2) Assigning

- 그 자리를 다른 숫자로 **assign**하고 싶다.

A(: , 2) = [10 ; 11 ; 12] ← 2열 전체가 10, 11, 12로 바뀌게 된다.

A =

```
    1    10
    3    11
    5    12
```

(1-3) append하고 싶다

A = [A , [100 ; 101 ; 102]] → column을 옆에다가 붙인거다!!! ([ ]로 표시하면 된다.)

➔ , (쉼표) 랑 ' ' (띄어쓰기)랑 같은 것이다. 나란히 붙이기!!!

A =

```
     1     10    100
     3     11    101
     5     12    102
```

- 만약에 row를 append하고 싶으면

```
>> A = [A ; [200 201 202]]
A =

     1     10    100
     3     11    101
     5     12    102
   200    201    202
```

(1-4) 두 개의 matrix를 붙이고 싶다.

```
>> A = [1 2; 3 4; 5 6]:
>>B = [11 12; 13 14 ; 15 16
>> C= [A B]
C =

    1     2    11    12
    3     4    13    14
    5     6    15    16

>> C = [A ; B]
C =

    1     2
```

3 4
5 6
11 12
13 14
15 16

(2) 행렬 만드는 다른 방식들 (ones, zeros, eye, rand, randn, magic)

>> v = 1 : 0.1 : 2 ← 1부터 0.1 step으로 2까지 (2포함) : range랑 비슷함

→ 열벡터를 만들어준다. 1 1.1 1.2 .... 1.9 2.0

```
>> v = 1:6
v =

   1   2   3   4   5   6
```

>> ones(2,3) → 1로 된 2x3의 matrix를 만들어준다.

```
ans =

   1   1   1
   1   1   1
```

```
>> 2*ones(2,3)
ans =

   2   2   2
   2   2   2
```

```
>> zeros(2,3)
ans =

   0   0   0
   0   0   0
```

```
>> eye (3,3)
ans =

Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1
```

```
>> eye(3)
```

ans =

Diagonal Matrix

1 0 0
0 1 0
0 0 1

>> rand(3,3) ← 0부터 1까지 랜덤한 숫자 3x3로!!
ans =

0.95038 0.69605 0.61569
0.70962 0.30824 0.75617
0.89219 0.58523 0.13924

>> randn(2,3) ← 가우씨안 distribution에서 2x3 랜덤 matrix (mean=0, std=1)
ans =

-0.91661 0.16785 0.72714
0.57808 0.47086 0.80971

>> A = magic (3) → 3 by 3 matrix인데 가로 세로 합이 일정한 행렬을 만들어준다.
A =

8 1 6
3 5 7
4 9 2

cf) **help**
help eye
help rand
help help

1.2. How to move data around

load
save the result

(1) Matrix size

(1-1) size 함수

A = [1 2 ; 3 4; 5 6]
size(A)

sz = size(A)

>> sz = size (A)

sz =

   3   2

➔ sz도 함수가 되었다.

size(A, 1) → size 함수의 결과값의 1번째 index가져와라 → row의 개수

>> size(A, 1)

ans = 3

size(A, 2) → size함수 결과의 2번째 index가져와라 → A행렬의 column의 개수

(1-2) length 함수 → 가장 긴 dimension을 가져다 준다.

>> v=[1 2 3 4]

v =

   1   2   3   4

>> **length**(v)

ans = 4

(2) 데이터 저장 및 불러오기

(2-1) pwd

어디에 있는지

(2-2) cd

change directory

(2-3) ls ← 리눅스랑 같네

그 directory에 있는 거 다 나오게

(2-4) data loading

load featuresX.dat

아니면

load ('featuresX.dat')

다음에

>> featuresX

>> size(featuresX)

없애려면

>> **clear** featuresX

>> **clear** ➔ 모든 변수를 다 없앤다.

앞의 10개의 data만 변수로 가져오려면

v = featuresX(1:10)

(2-5) who

who 하면 current data에서 어떤 변수들이 정의됐는지 볼 수 있다.

>> who

Variables in the current scope:

A ans sz v w

>> whos ← 대박!!!! 어떤 data인지, matrix인지

(2-6) 저장 방법

>> save hello.mat v;

>> save hello.txt v -ascii % save as text(ASCII)

1.3. Matrix computing

(1) Matrix multiplication

>> A * C

ans =

5 5
11 11
17 17

(2) 각 원소별로 연산하기

>> A . * B → 각 원소끼리 곱하는 것이다. 엄청 특이하네. A의 1,1을 B의 1,1과 곱하고, A의 1,2를 B의 1,2와 곱하는 것이다.

>> A . ^2 → element wise squaring

>> 역수를 취하고 싶을 때

v = [1 ; 2 ; 3]

>> 1./v ← 1을 v의 각 원소 별로 나눠라.

ans =

1.00000

```
    0.50000
    0.33333

>> log(A)      ← 각 원소를!!
ans =

    0.00000   0.69315
    1.09861   1.38629
    1.60944   1.79176

>> exp(v)       ← e를 base로!!
ans =

     2.7183
     7.3891
    20.0855

>> abs(v)    ← 다 절대값으로

>> -v
ans =

   -1
   -2
   -3

>> 행렬 각 원소에 1을 더하고 싶을때
>> v + ones(length(v), 1)      ← v만큼의 열을 만들어서 1을 더해야 하니까 length!!!!
ans =

    2
    3
    4

>> v + 1      ← 같은 거임!!!
```

(3) Transpose, flip, inverse

(3-1) Transpose

```
>> A'
ans =

    1    3    5
    2    4    6
```

(3-2) flip up down

>> flipud(eye(9)) ← 바꿀 수 있음

ans =

Permutation Matrix

0 0 1
0 1 0
1 0 0

(3-3) Inverse

- inverse하고 싶으면

pinv(A) ← pseudo inverse에 해당한다.

(4) 행렬 내의 값 분석

a = [1 15 2 0.5]

(4-1) MAX 찾기 : 기본이 column단위

>> val = max(a) ← 최대값

val = 15

- index까지 알고 싶으면

>> [val, ind] = max(a)

val = 15

ind = 2

>> max (A)하면 → column wise maximum을 한다.

>> max (rand(3))

ans =

0.60958 0.93724 0.70481

→ 열 단위로!!

>> max (rand(3), rand(3)) → rand(3)와 rand(3) 두 개의 같은 index의 원소를 비교하여 3x3의 matrix를 보여줌

ans =

0.86301 0.80544 0.97582
0.90647 0.99723 0.33394
0.78273 0.23280 0.70618

>> max (A) → column wise maximum

>> max (A, [ ], 1) → column wise maximum
>> max (A, [], 2) → per row maximum

- 행렬 전체에서 가장 큰 원소를 찾고 싶으면
>> max(max(A))
아니면
>> max (A(:)) ← vector로 펼쳐놓은 다음에!!

(4-2) 부등호 비교

- 값 비교를 하고 싶으면
>> a < 3

>> a
a =

    1.00000   15.00000   2.00000   0.50000

>> a < 3
ans =

  1 0 1 1 ← 2번째 원소는 아니구나

>> **find** (a < 3) ← 3보다 작은 원소의 index를 알려준다.

>> find(a<3)
ans =

   1   3   4

>> [r, c] = find (A >= 7) → 이렇게 해야 원소 index를 알 수 있음. 그냥 vector처럼 1자리면 안됨
r =

   1
   3
   2
c =
   1
   2
   3

(4-3) 합 : 기본이 column별

>> sum (a)
ans = 18.500

- column 별로 sum하려면

>> sum(A, 1)

- row 별로 sum하려면

>> sum(A, 2)

- 만약에 행렬의 대각선을 곱하고 싶으면

A = magic (9)

A . *eye(9) ← 원소별로 곱하니까 E 대각행렬의 1과 곱해진 부분만 남게 된다.

ans =

47	0	0	0	0	0	0	0	0
0	68	0	0	0	0	0	0	0
0	0	8	0	0	0	0	0	0
0	0	0	20	0	0	0	0	0
0	0	0	0	41	0	0	0	0
0	0	0	0	0	62	0	0	0
0	0	0	0	0	0	74	0	0
0	0	0	0	0	0	0	14	0
0	0	0	0	0	0	0	0	35

다 더하고 싶은 것이니까

>> sum(sum(A.*eye(9)))

(4-4) 곱

>> **prod** (a) → 다 곱해!!

(4-5) 올림 내림

>> **ceil** (a) → 올림

>> **floor** (a) → 내림

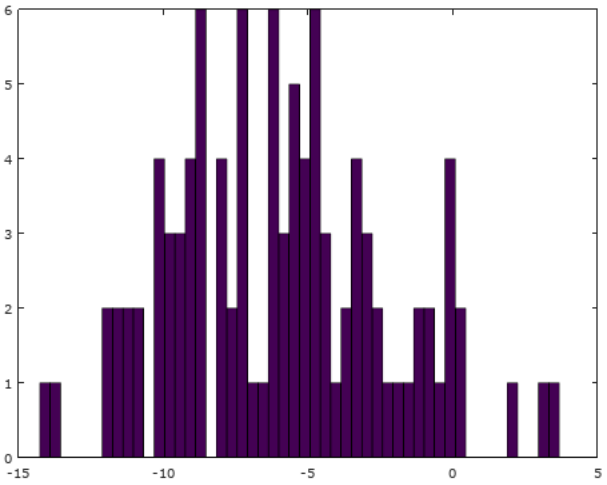
1.4. Plotting

(0) 히스토그램 만들기

>> w = -6 + sqrt(10)*randn(1,100);

>> hist(w)

>> hist(w,50) ← 50 구간을 만들자.

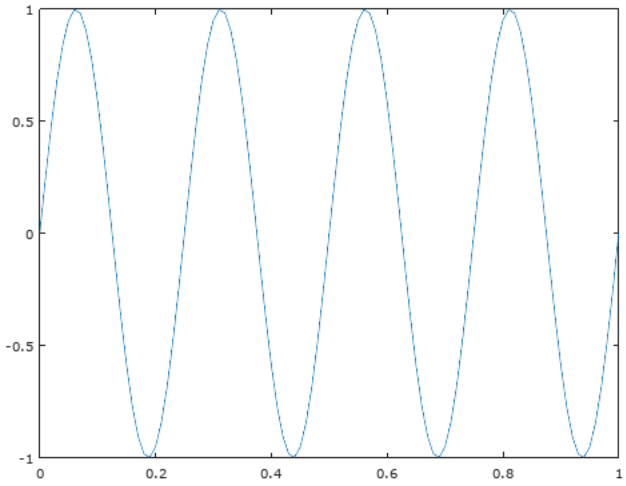


(1) Plot 함수 : (x 축, y 축)으로 하면 된다.

>> t= [0 : 0.01 : 1];

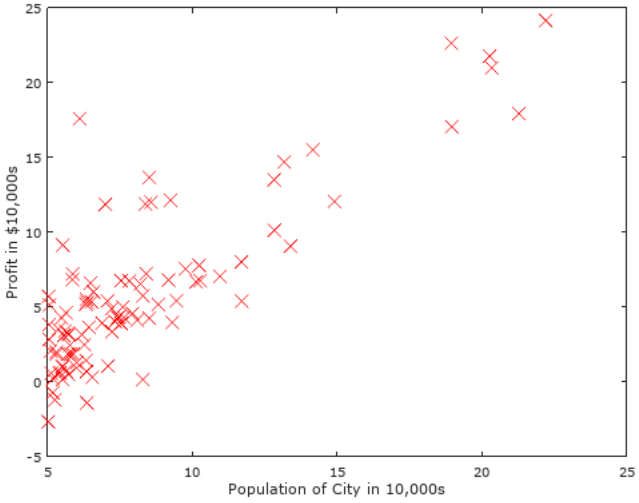
>> y1 = sin (2*pi*4*t);

>> plot(t, y1)

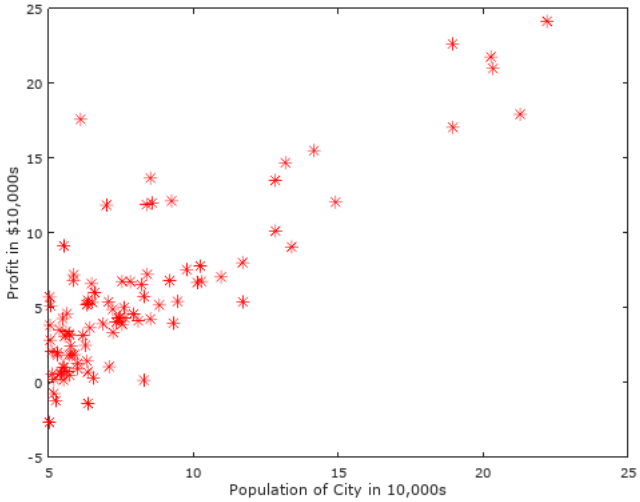


>> plot(x, y, 'rx', 'MarkerSize', 10);

← rx라는 것은 red x 표시를 하겠다는 뜻이다. MarkerSize는 x의 크기를 의미한다.



cf) r*를 하면 이렇게 된다.

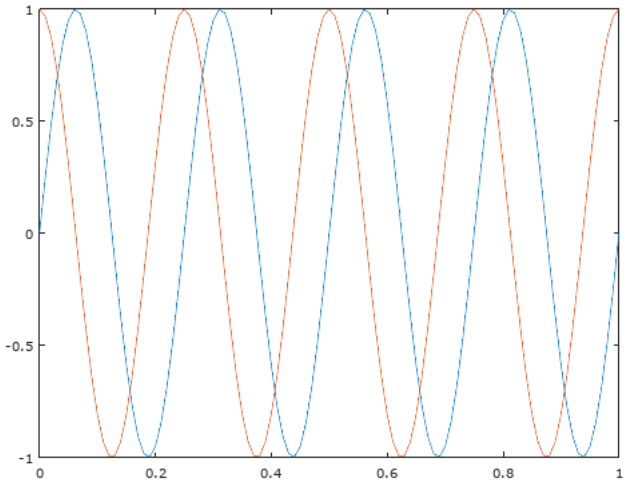


- 위에 고정시키는 방법

>> **hold on**;

>> y2 = cos(2*pi*4*t);

>> plot (t,y2)



- plot하는 방법 심화 (2주차 octave 과제)

(2) labeling!!!

>> **xlabel**('time')

>> **ylabel**('value')

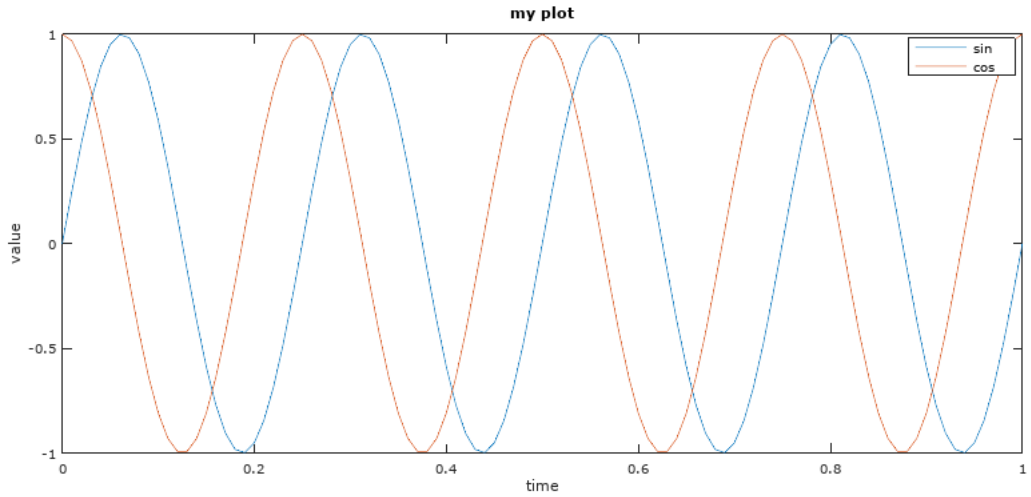
>> **legend** ('sin', 'cos')

>> **title** ('my plot')

- range 바꾸기

>> axis ([0.5 1 -1 1]) ➔ 처음 2개는 x축이 어디서부터 어디까지, 뒤에 두 개는 y축이 어디서부터 어디까지

>> **clf** ; ➔ clear figure



(3) 저장하려면

>> cd 'C:₩Users₩owner₩Desktop'; print -dpng 'myPlot.png'

>> close

figure 없어짐

(4) 따로따로 띄우려면

>> figure (1) ; plot (t, y1);

>> figure (2) ; plot (t, y2);

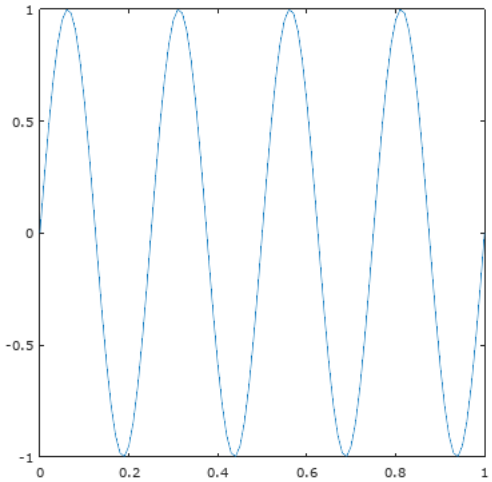
- 1개의 창에 같이 띄우려면

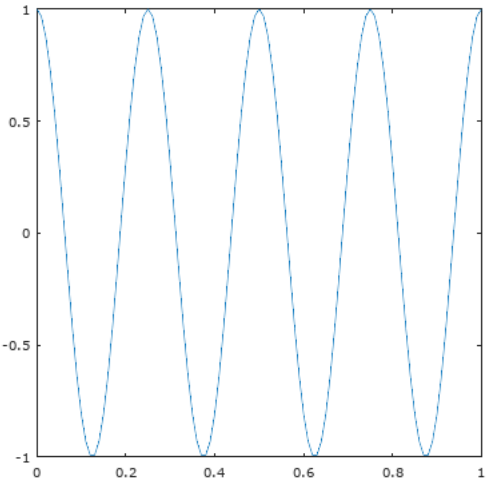
>> **subplot(1,2, 1)** → 1 x 2 grid를 만들고 첫번째 element에 access

>> plot(t, y1);

>> subplot(1, 2, 2) ;

>> plot (t, y2);





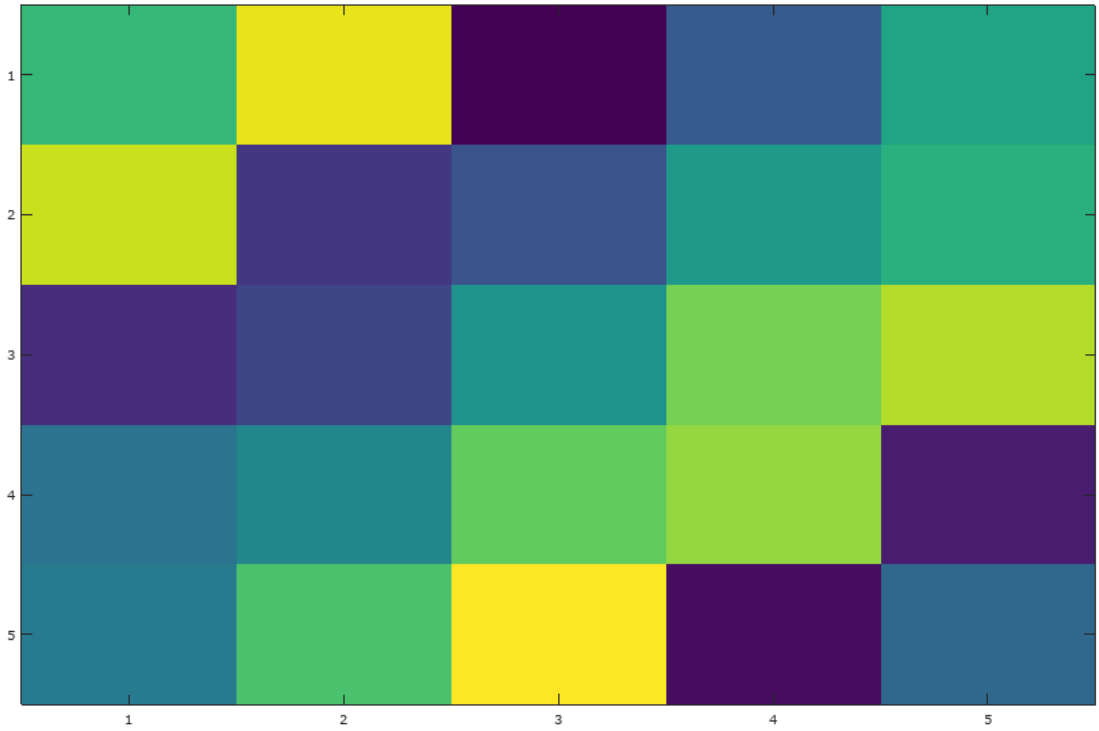
(5) heat map plotting하기

>> A = magic(5)

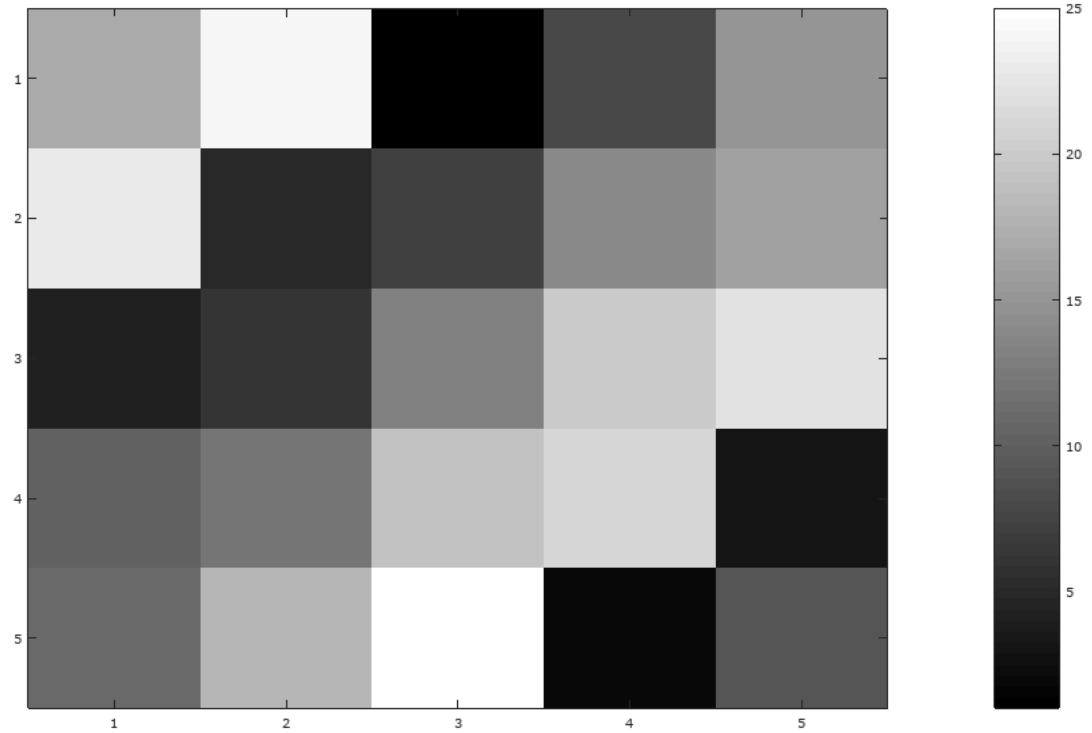
A =

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

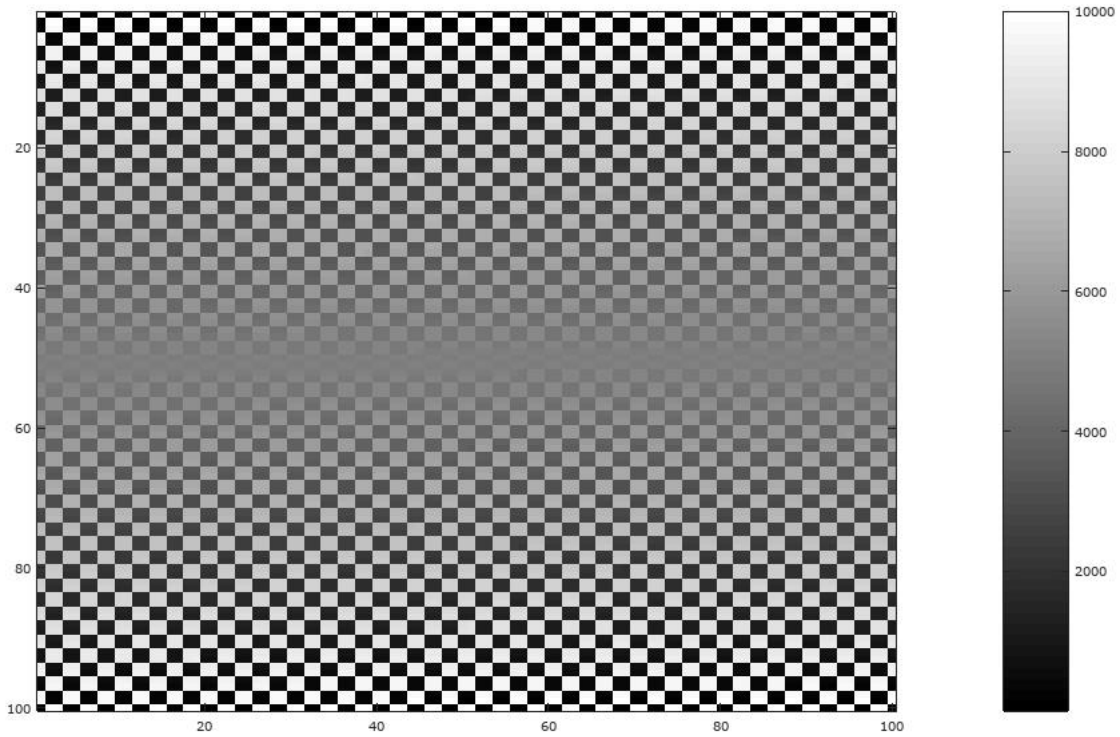
>> **imagesc**(A)



>> **imagesc(A), colorbar, colormap gray**; ➔ comma chaining of commands (연결해서 쓰는 거다)



>> imagesc(magic(100)), colorbar, colormap gray;



1.5. Control statements

(1) v (i) = ➔ 이걸로 처리하면 된다!!

- 빈공간 만들기

>> **v = zeros(10, 1)**

>> for i=1:10, ← 1부터 10까지

```
>> for i=1:10,
v(i) = 2^i;
end;

>> v
v =

     2
     4
     8
    16
    32
    64
   128
   256
   512
  1024

>> i = 1;
```

```
>> while i <= 5,
v(i) = 100;
i=i+1;
end;
>> v
v =

     100
     100
     100
     100
     100
      64
     128
     256
     512
    1024
```

- break 사용하는 방법

```
i = 1;
while true,
   v(i) = 999;
   i = i+1;
   if i == 6,
      break;
   end;
end;
```

처음 5개는 999로 바뀌게 된다.

```
>> v(1)
ans = 999
```

- if statement

```
if v(1) == 1,
   disp('the value is one');
elseif v(1) ==2,
   disp('the value is two');
else,
   disp('haha');
end;
```

➔ elif만 다른 듯. 끝에 항상 ;을 넣는다는 거랑.

- function을 사용해보자.

pwd해서 있는 directory에

function 이름.m
으로 저장하면 된다!!

그 다음에 바로 squareThisNumber(5) 하면 적용된다. → .m 으로 저장한다는 것이 핵심포인트.

- search path 사용하기

>> **addpath**('C:₩Users₩Owner₩Desktop') 이렇게 해놓으면 다른 directory에 있어도 여기있는 함수를 찾아보게 된다.

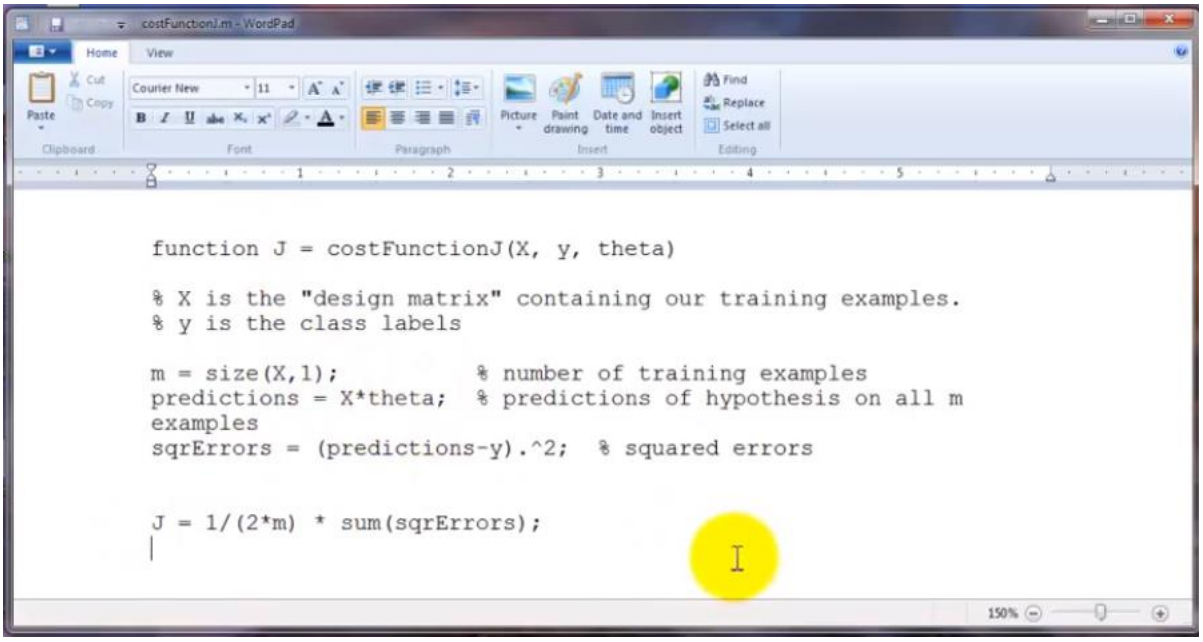
- octave는 많은 value를 return할 수 있다.

function [y1, y2] = squareAndCubeThisNumber(x)
y1 = x ^2;
y2 = x ^3;

>> [a, b] = squareAndCubeThisNumber(x)
>>a
>>b
다른 게 나오게 된다.

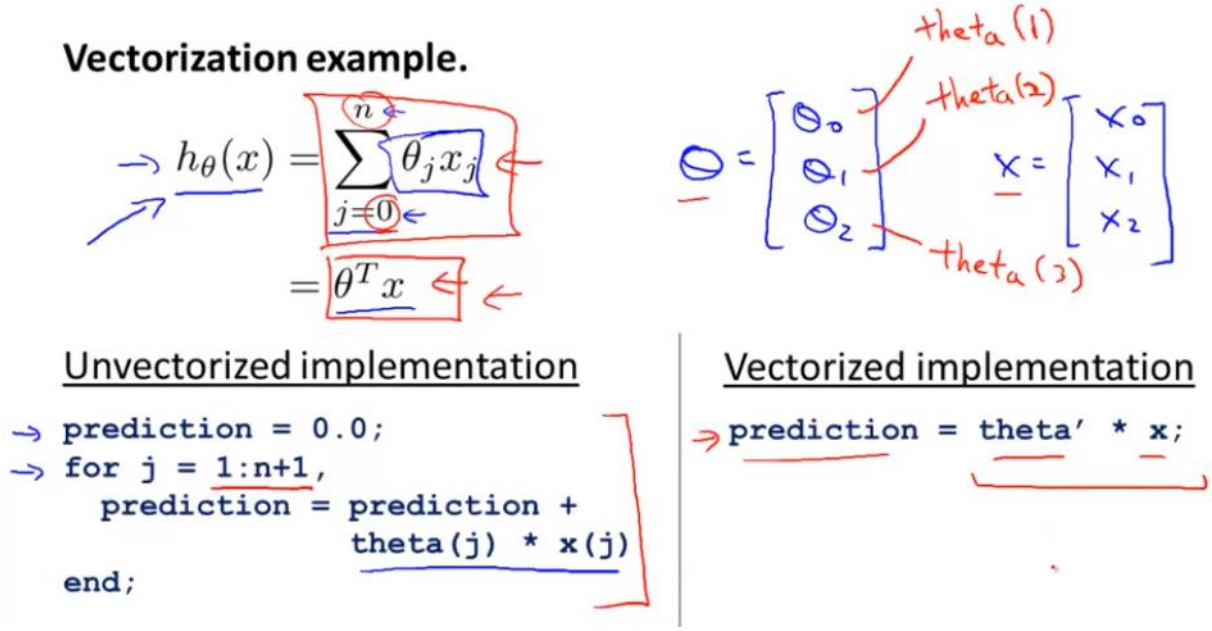
- 종합
→ Cost function

```
>> X = [1 1; 1 2; 1 3]
X =

   1   1
   1   2
   1   3

>> y = [1; 2; 3]
y =

   1
   2
   3

>> theta = [0;1];
```

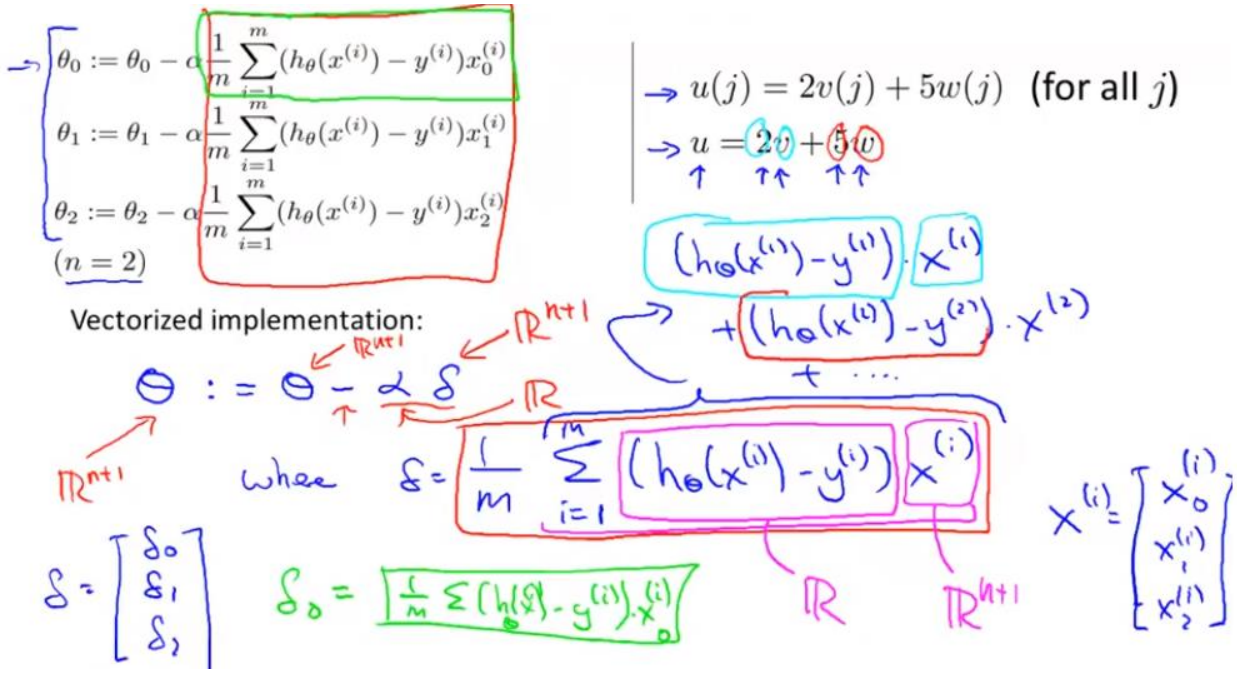


1.6. Vectorization

- for loop 보다는 dot product를 하면 한 줄 만에 계산할 수 있다!!



- Gradient descent 도 vectorized implementation을 할 수 있다. 식을 간단히 써서 θ :=θ - αδ. 이거랑 같은 것이다. For loop은 가급적 지양하자. 굉장히 오래 걸릴 수 있다!!



- 실전 적용 (Octave)

Gradient descent 하는 법

<computeCost.m>

```
function J = computeCost(X, y, theta)
J = 0;
J = 1/(2*m) * sum(((X * theta) - y).^2)
end
```

```
X = [ones(m, 1), data(:,1)]; % Add a column of ones to x
theta = zeros(2, 1); % initialize fitting parameters

% Some gradient descent settings
iterations = 1500;
alpha = 0.01;

fprintf('₩nTesting the cost function ...₩n')
% compute and display initial cost
J = computeCost(X, y, theta);
fprintf('With theta = [0 ; 0]₩nCost computed = %f₩n', J);
```

- 미분을 하는 방법

디렉토리에서 sympy 다운로드하기. (https://github.com/cbm755/octsympy)

* 버그 해결 : https://www.nntp.perl.org/group/perl.modules/2010/01/msg69392.html

Download the symbolic-win-py-bundle-2.5.0.zip file from releases.

Start Octave

At the Octave prompt, type pkg install symbolic-win-py-bundle-2.5.0.zip.

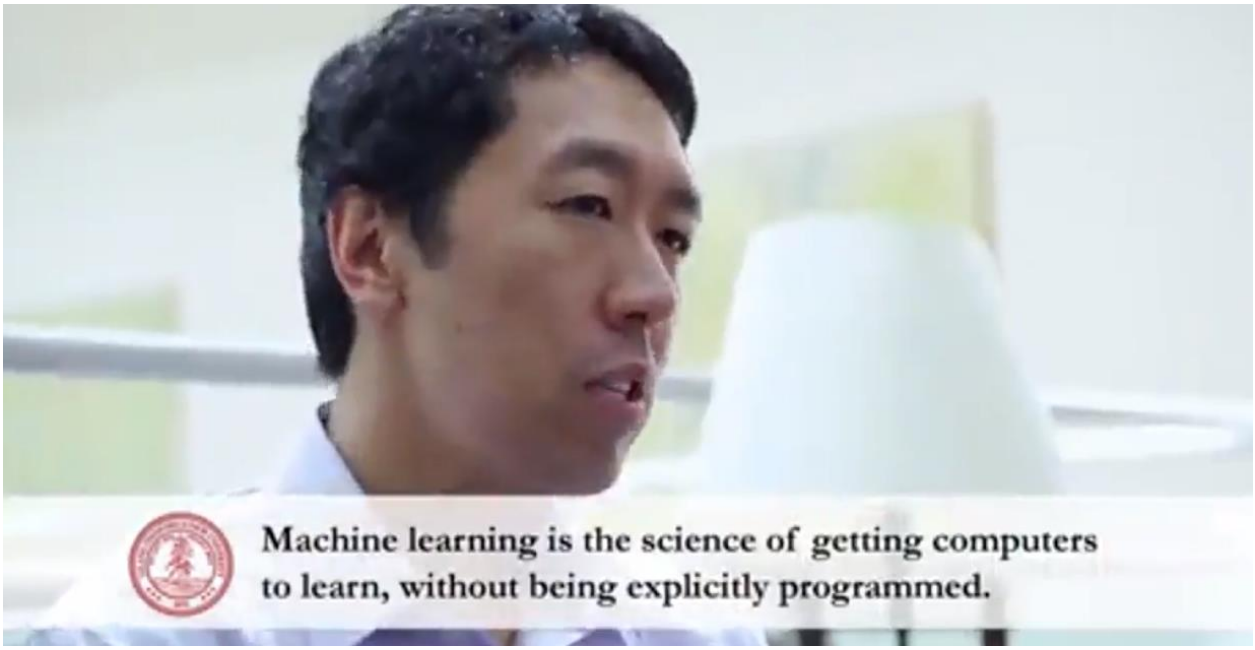
At the Octave prompt, type pkg load symbolic.

At the Octave prompt, type syms x, then f = (sin(x/2))^3, diff(f, x), etc.

→ 버그는 해결했는데 syms x 가 안되네 여전히.. (18. 3. 11)

II. Machine learning

0. What is machine learning!



0.1. 왜 필요한지? (동기부여 뿜뿜)

Examples:

- Database mining
  - Large datasets from growth of automation/web.
  - E.g., Web click data, medical records, biology, engineering
- Applications can't program by hand.
  - E.g., Autonomous helicopter, handwriting recognition, most of Natural Language Processing (NLP), Computer Vision.
- Self-customizing programs
  - E.g., Amazon, Netflix product recommendations
- Understanding human learning (brain, real AI).

0.2. Machine learning의 정의

- Field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel(1959)

- 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.

→ P는 승률을 의미할 수도

- 예시 :

"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. T
- ◯ Watching you label emails as spam or not spam. E
- ◯ The number (or fraction) of emails correctly classified as spam/not spam.
- ◯ None of the above—this is not a machine learning problem. P

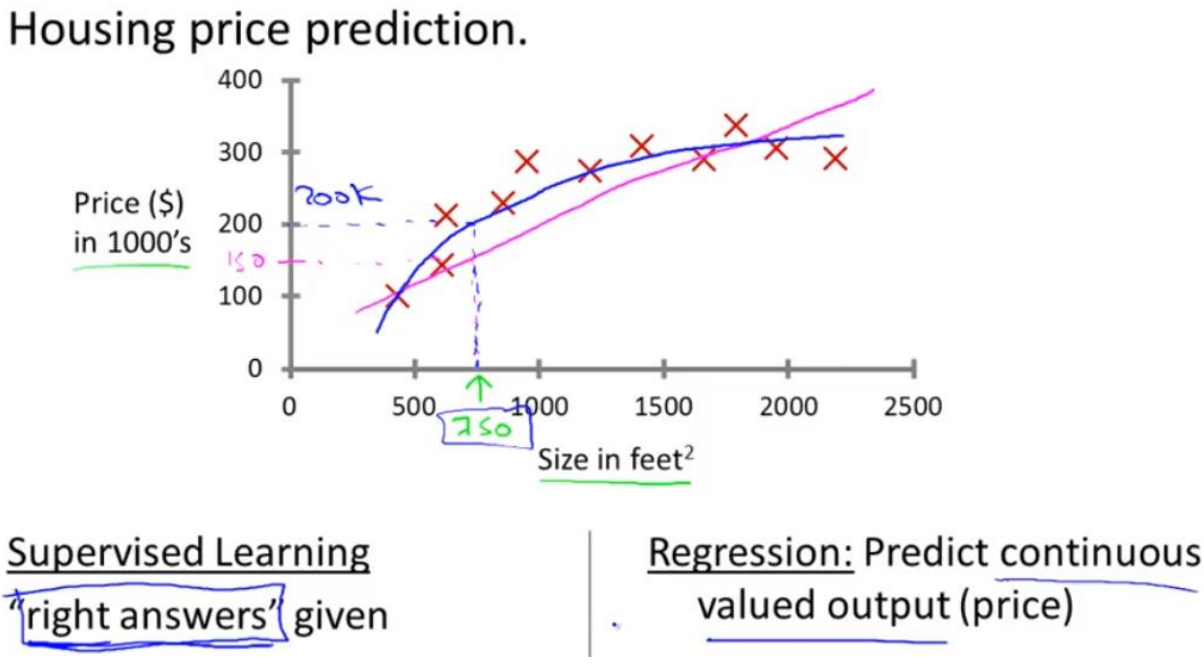
0.3. 머신러닝의 종류는 무엇인가?

(1) Supervised learning

- we gave the "right answers" ➔ It can be a **regression problem**(continuous) or a **classification problem**(discrete).

→ 즉, 데이터 셋을 먼저 주는 것이 핵심이다. 내가 알고 있었던 machine learning의 종류인 것 같다!

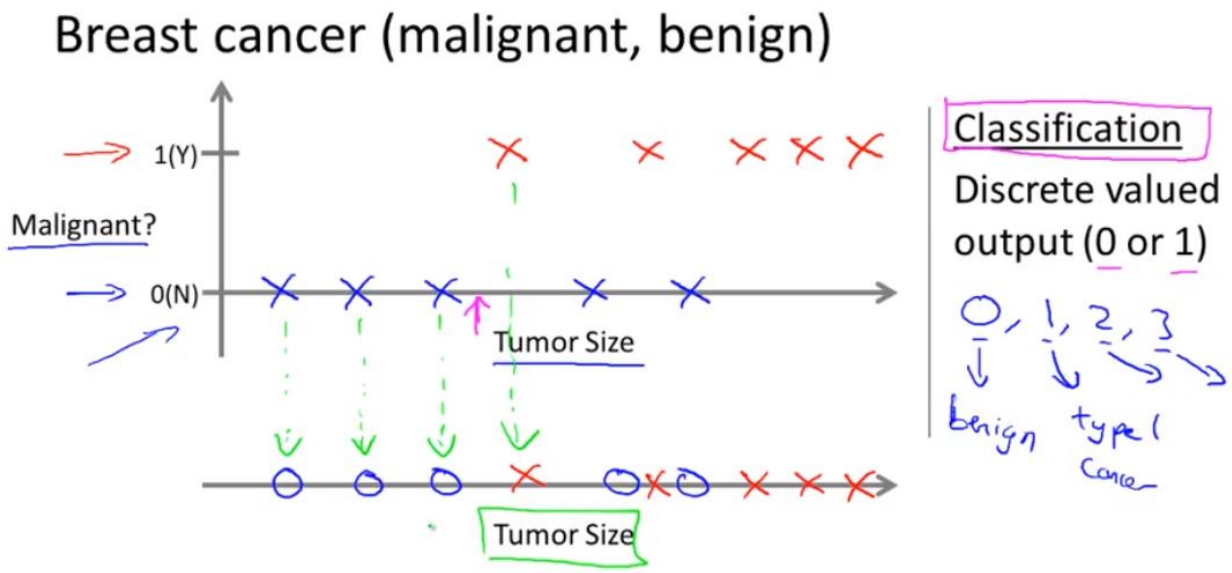
(1-1) regression problem :



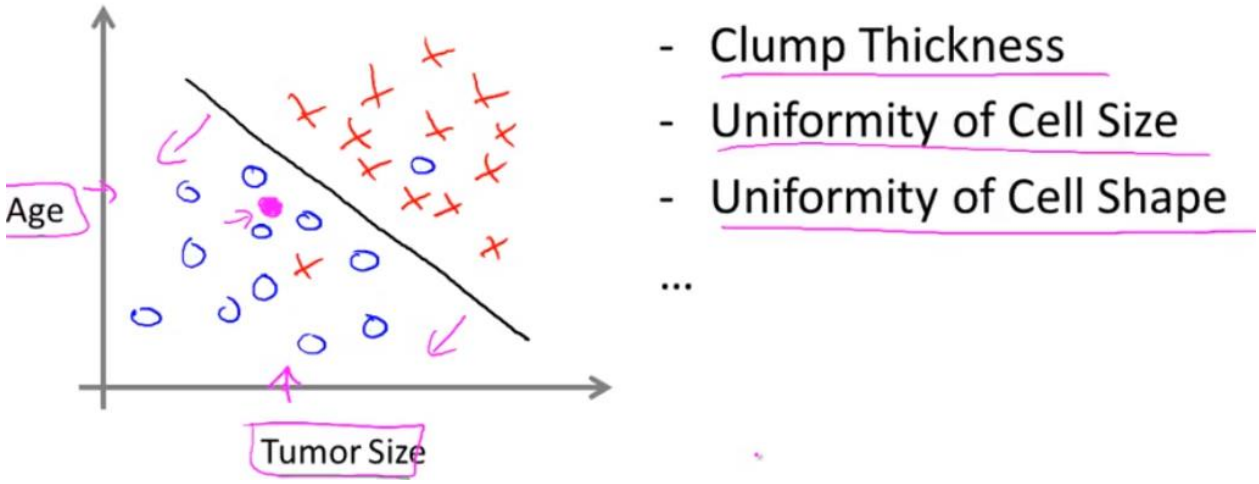
→ machine learning question is "how much is the 750 square feet house sold?"

- example 2 : Regression - Given a picture of a person, we have to predict their age on the basis of the given picture

(1-2) classification problem : 여기서의 classification은 종속변수가 descret하다. 0, 1 뿐만 아니라, 0, 1, 2, 3이어도 괜찮다. → logistic regression



→ 0, 1을 다른 기호로 처리해서 1차원으로 표시할 수도 있다. 그렇게 하면 tumor size말고 다른 독립변수도 plotting할 수 있게 된다. 2차원으로 하면. Machine learning의 목적은 age, tumor size를 알 때, 과연 그 사람의 tumor가 benign한지, malignant한지를 알아내는 것이다. 그것을 위해서는 그 경계에 선을 그릴 수 있어야 한다. 따라서 그것이 바로 머신러닝의 작업이 된다. 만약에 변수가 2개가 아니라 무한대로 늘어나게 된다면? 그것도 처리할 수 있다.

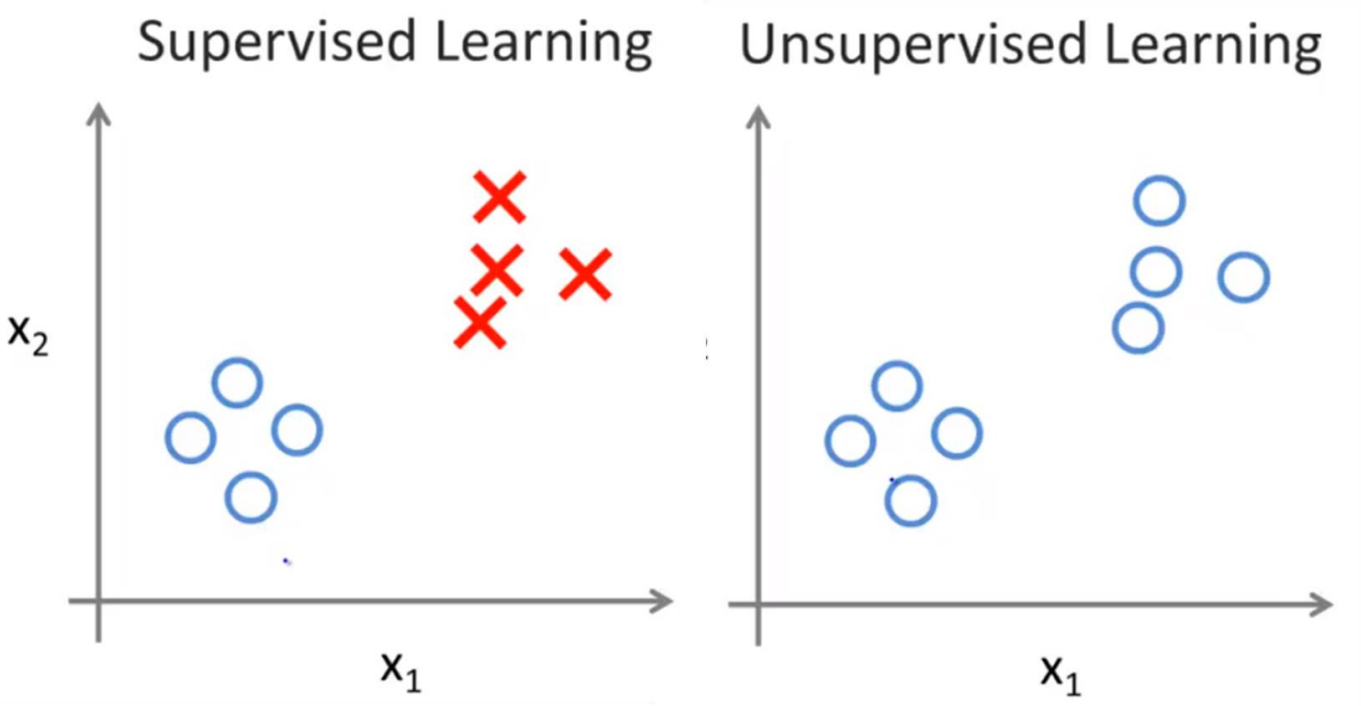


- example 2 :
Given data about the size of houses on the real estate market, try to predict their price. Price as a function of size is a continuous output, so this is a regression problem.

We could turn this example into a classification problem by instead making our output about whether the house "sells for more or less than the asking price." Here we are classifying the houses based on price into two discrete categories.

(2) Unsupervised learning

- 정의 : data의 label이 다르다!! : unsupervised learning은 answer를 주지 않은 채 structure를 찾아보라고 하는 것이다. Not labeled. 즉, we have no or little idea what our results should look like.
➔ "We can derive structure from data where we don't necessarily know the effect of the variables."
- 목적 : Data set을 받고 find some structure in this data set.

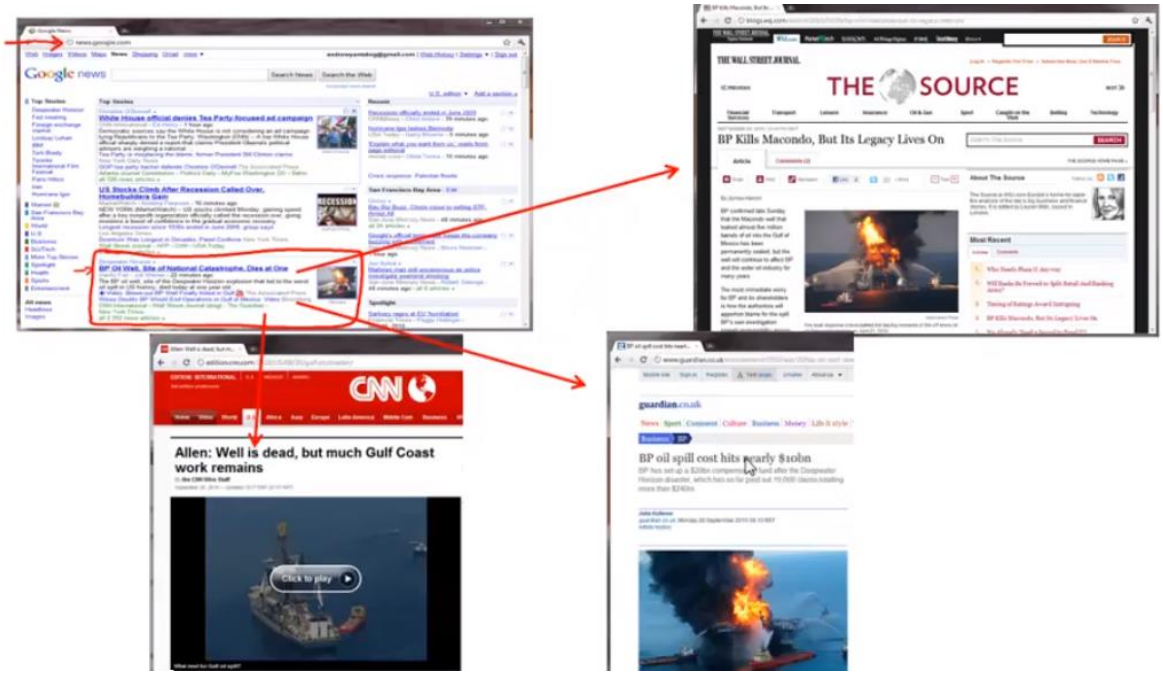


- 차이점에 대한 심화된 설명 : **When you already have the answer or the answer is already given into the data set then it's Supervised learning.** The input data has correct output. Like House price, you already know the price of some area(sq.ft). So your input data (data set you have) already have correct output. Or like if the cancer is malignant or not? You already have a data set with correct answer.

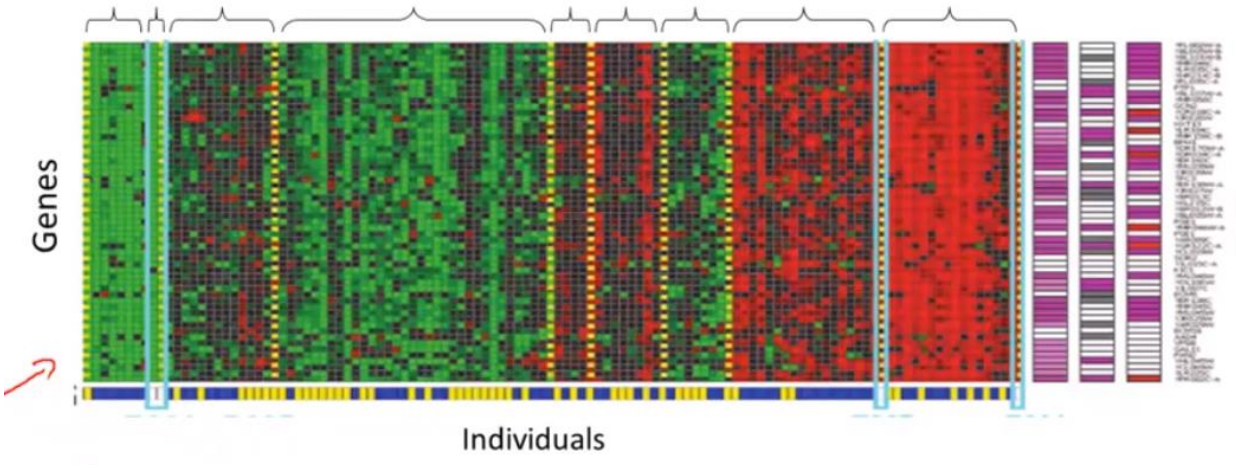
When you have data set and don't know the answer then it's Unsupervised learning. Like you've bunch of photos of several people but you don't know which is which. So you categories them with which ones are alike. You don't have the correct answer here. You categorise them based on similarities. I hope you get the difference. Cheers!

(2-1) Clustering algorithm : 두 cluster로 나눌 수 있다.

- example 1 : News를 clustering한다. 이게 뭔지는 알려주지 않았는데도..

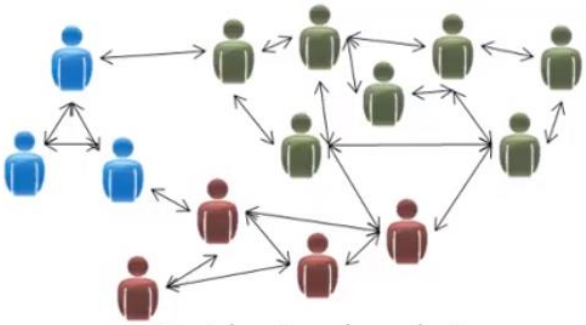


- example 2 : 딱히 먼저 분류해놓은 것은 아닌데, 유전자 발현 여부의 패턴에 따라서 분류를 해놨다. cluster로!



- 다른 예 :

Organize computing clusters



Social network analysis

Market segmentation

Astronomical data analysis

- 예시 문제 :

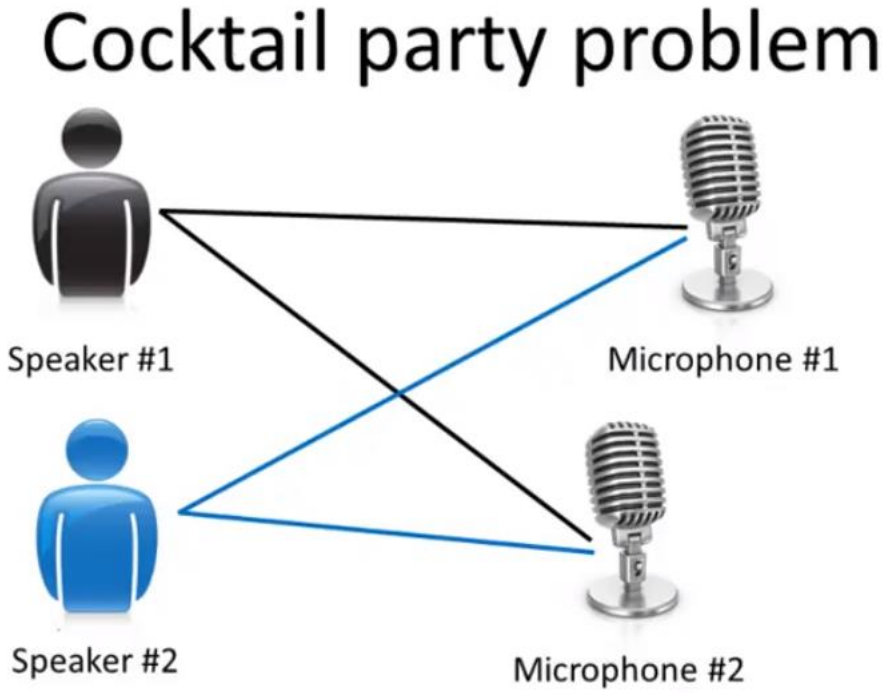
Of the following examples, which would you address using an unsupervised learning algorithm? (Check all that apply.)

- ☐ Given email labeled as spam/not spam, learn a spam filter.
- ☐ Given a set of news articles found on the web, group them into set of articles about the same story.
- ☐ Given a database of customer data, automatically discover market segments and group customers into different market segments.
- ☐ Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.

(2-2) Non-clustering algorithm : "Cocktail Party Algorithm"

- Non-clustering: The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a cocktail party)

→ Cocktail party problem : 파티에 가면 잘 안 들려!



→ 와! 두 개의 목소리를 나눌 수 있다. 거리로 구분을 했었다. 거리에 따라서 한 쪽에서는 크게, 한쪽에서는 작게 들릴 테니.

```
[W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

→ 이 것은 octave로 한 줄로 이 알고리즘을 짤 수 있다. python으로 하면 여러 줄로 짜야 한다. Octave은 prototyping을 할 수 있다. 이걸 하면 더 빨리 이해할 수 있고, 더 잘 구동시킬 수 있다. 많은 회사들에서도 octave를 통해서 prototype를 만들어보고 그 다음에 잘 되면 python, java로 이행한다.

Supervised learning

1. Supervised learning - Model representation

1.1. Supervised learning의 이해

1.1.1. Notation :

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

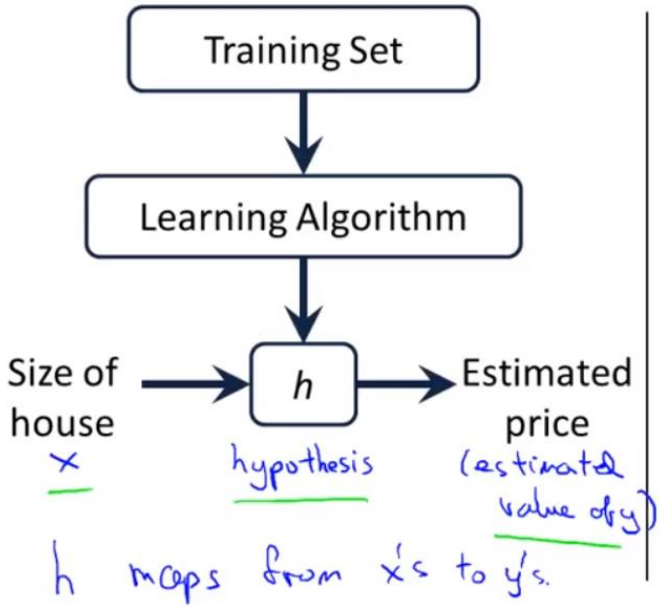
Notation:
- **m** = Number of training examples
- **x**'s = "input" variable / features
- **y**'s = "output" variable / "target" variable

(x, y) - one training example

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

1.1.2. h (hypothesis)

: hypothesis는 독립, 종속변수를 잇는 함수이다! To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X → Y so that h(x) is a "good" predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis.



How do we represent *h* ?

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

Shorthand: $h(x)$

→ 이 경우에는 univariate linear regression이다. 변수가 하나인 경우!!

1.1.3. $J(\theta_0, \theta_1)$ (cost function)

Training Set

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

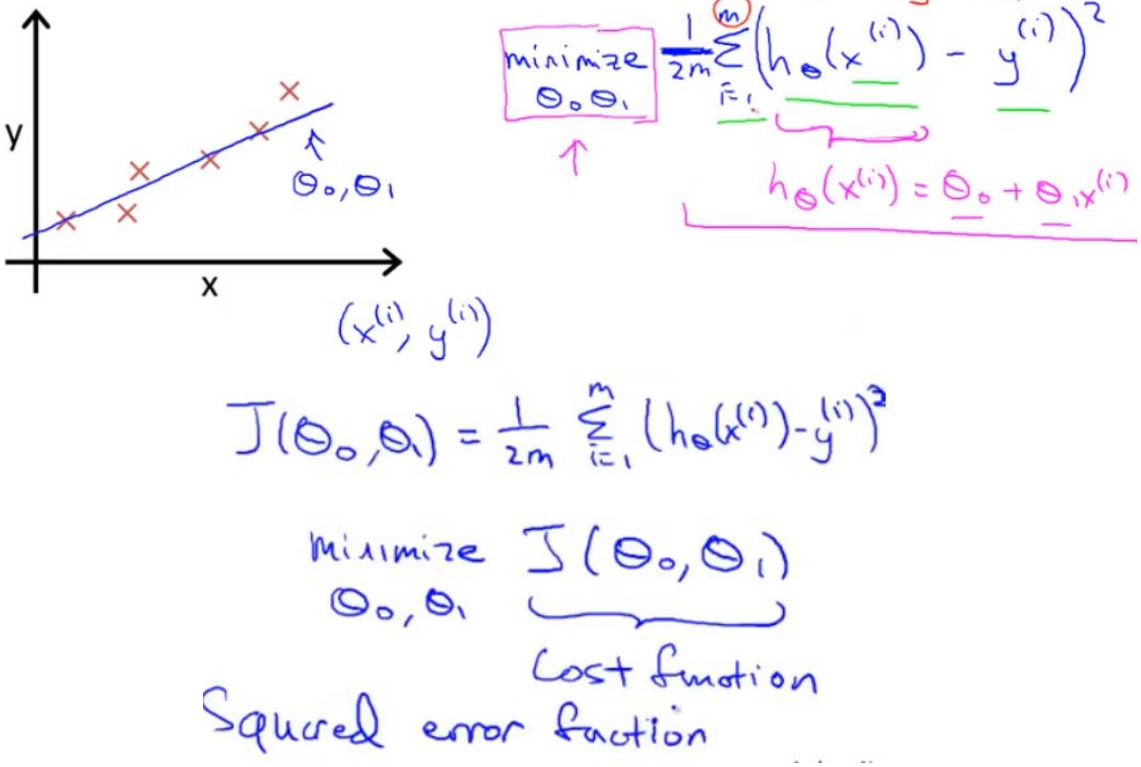
m = 47

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

θ_i's: Parameters

How to choose θ_i's ?

- 목적 : How do we choose "hypothesis"? We use cost function (square error function)
- 방법 : Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)



- 언어 (이해 못하고 있음 → 이해함!!) :

$$J(₩theta_0, ₩theta_1) = ₩dfrac {1}{2m} ₩displaystyle ₩sum_{i=1}^m ₩left (₩hat{y}_{i}- y_{i} ₩right)^2 = ₩dfrac {1}{2m} ₩displaystyle ₩sum _{i=1}^m ₩left (h_₩theta (x_{i}) - y_{i} ₩right)^2$$

→ To break it apart, it is $$₩frac{1}{2}$$ $$₩bar{x}$$ where $$₩bar{x}$$ is the mean of the squares of $$h_₩theta (x_{i}) - y_{i}$$, or the difference between the predicted value and the actual value.

→ 2로 다시 나누는 이유 : The mean is halved $$₩left(₩frac{1}{2}₩right)$$ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $$₩frac{1}{2}$$ term.

→ 제곱을 편미분하면 2가 내려오니까 2로 나눠버리면 편하잖아.

$f(x)^2 = 2*f(x)*f'(x)$

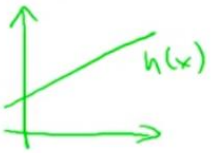
1.1.4. h, J와 목표(Goal)식 :

Hypothesis:

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

Parameters:

θ_0, θ_1



Cost Function:

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

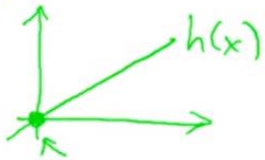
Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Simplified

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

$\theta_0 = 0$

θ_1



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

$\theta, x^{(i)}$

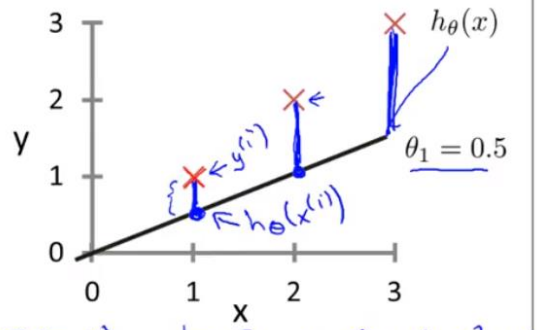
$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$

(1) Simplified version 실제 계산해보기 : θ_0=0이라고 가정

(1) J(0.5) = 0.58

$h_\theta(x)$

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

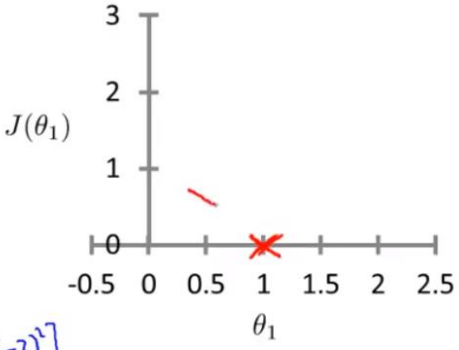


$J(0.5) = \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$

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

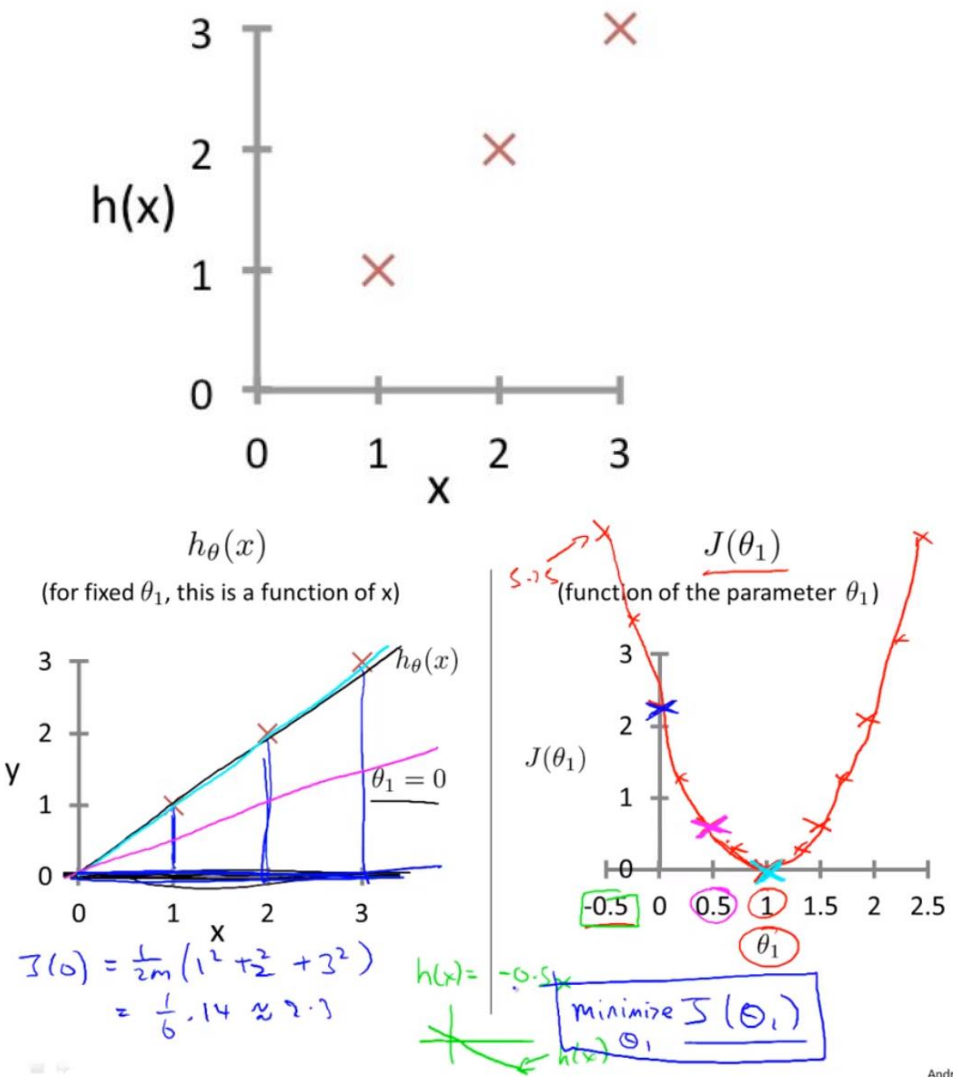
$J(\theta_1)$

(function of the parameter θ_1)



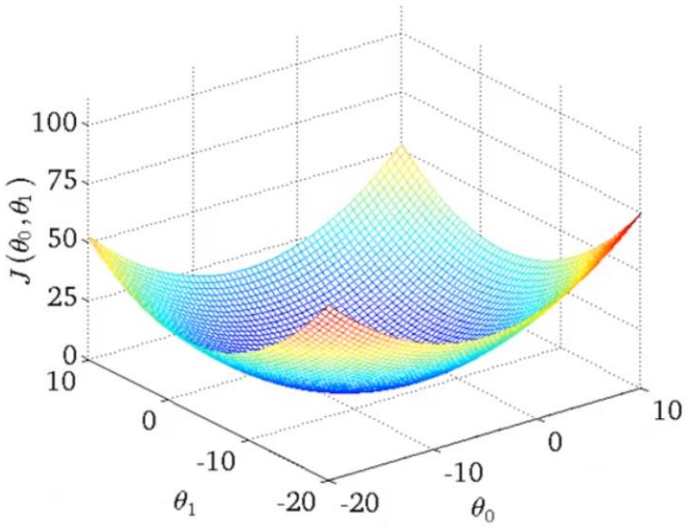
(2) J(0) = 14/6

Suppose we have a training set with m=3 examples, plotted below. Our hypothesis representation is $$h_\theta(x) = \theta_1x$$, with parameter $$\theta_1$$. The cost function $$J(\theta_1)$$ is $$J(\theta_1) = \frac{1}{2m} \sum^m_{i=1} (h_\theta (x^{(i)}) - y^{(i)})^2$$. What is $$J(0)$$?

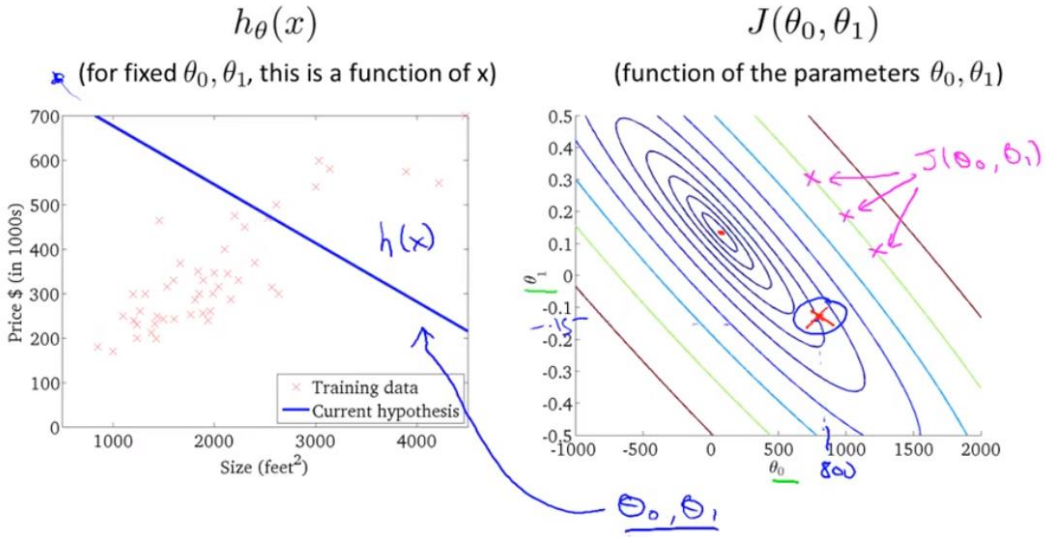


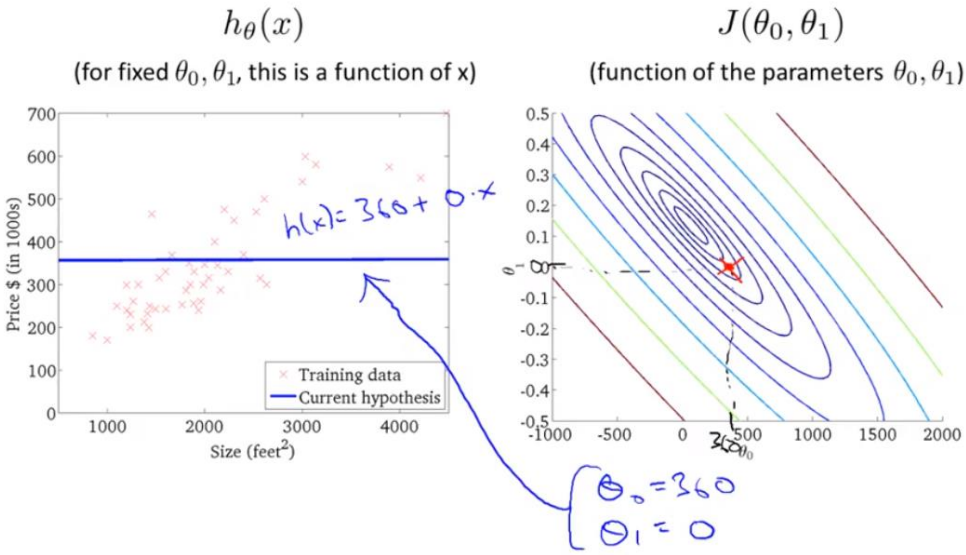
→ 아하! 이런 식으로 J(0), J(1), J(0.5)를 해가다 보면 포물선이 그려지고, 결국에는 J(1)이 최선이라는 것을 알 수 있구나!. θ_1이 1일 때가 Minimize J를 할 수 있는 것이다.

(2) θ1과 θ0이 다 살아있을 때의 cost function



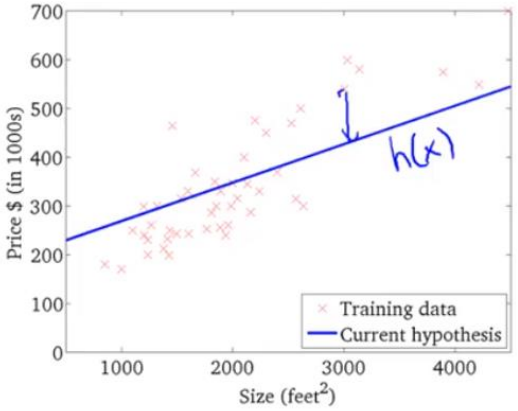
→ **Contour figure**





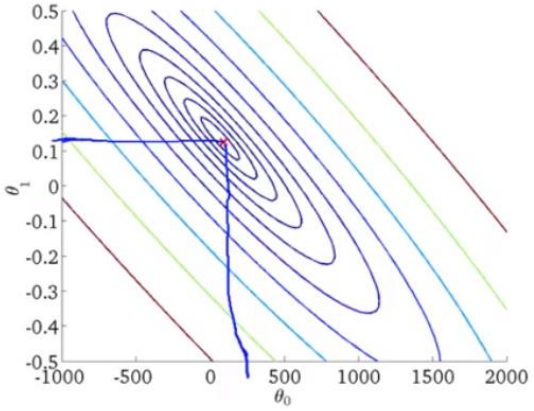
$h_\theta(x)$

(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)



→ 나중에는 θ0, θ1이 아니라 더 많아져서 θ2, 등등 이 생기게 되면, 이것은 visualization이 힘들어지게 된다.

Supervised machine learning : Cost function minimize method

➔ J(Cost function) minimize method : Gradient descent, Normal equation

1.2. J를 minimize하는 알고리즘-1 : gradient descent

1.2.1. Gradient descent 정의 : minimize를 하는 알고리즘. 사실 J가 어떤 것이든 폭 넓게 적용할 수 있다.

Have some function $J(\theta_0, \theta_1)$ $J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

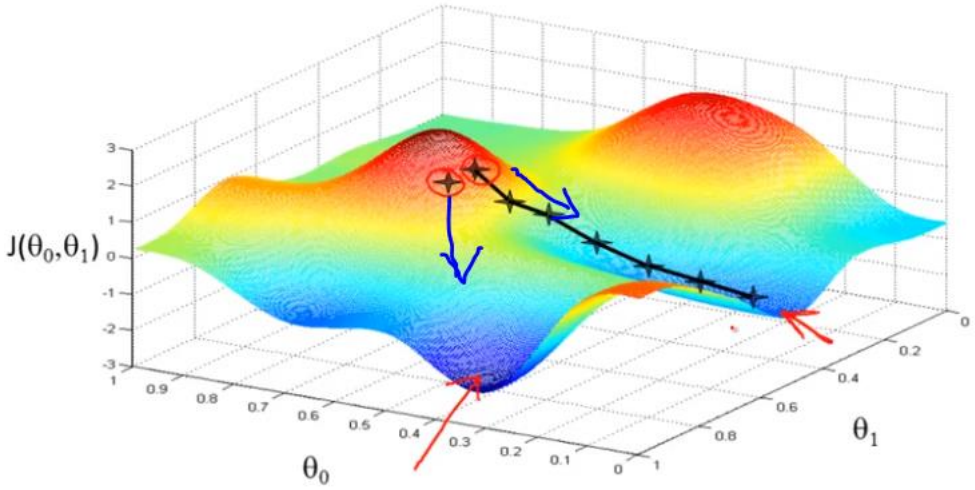
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$ $\min_{\theta_0 \cdots \theta_n} J(\theta_0, \ldots, \theta_n)$

Outline:

- Start with some θ_0, θ_1 (Say $\theta_0 = 0, \theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

  until we hopefully end up at a minimum

- 직관 : 한 점에서 시작한다. → 360도를 둘러보고, 어느 방향으로 babystep하면 더 빨리 내려갈 수 있을까?를 결정하고 나아간다. → babystep을 간 지점에서 다시 그렇게 한다.
- 특징 : 초기점이 조금만 달라져도 도달하는 local minimum이 달라질 수 있다.



1.2.2. 알고리즘 : We make steps down the cost function in the direction with the steepest descent.

repeat until convergence {

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

}

learning rate

Repeat until convergence:

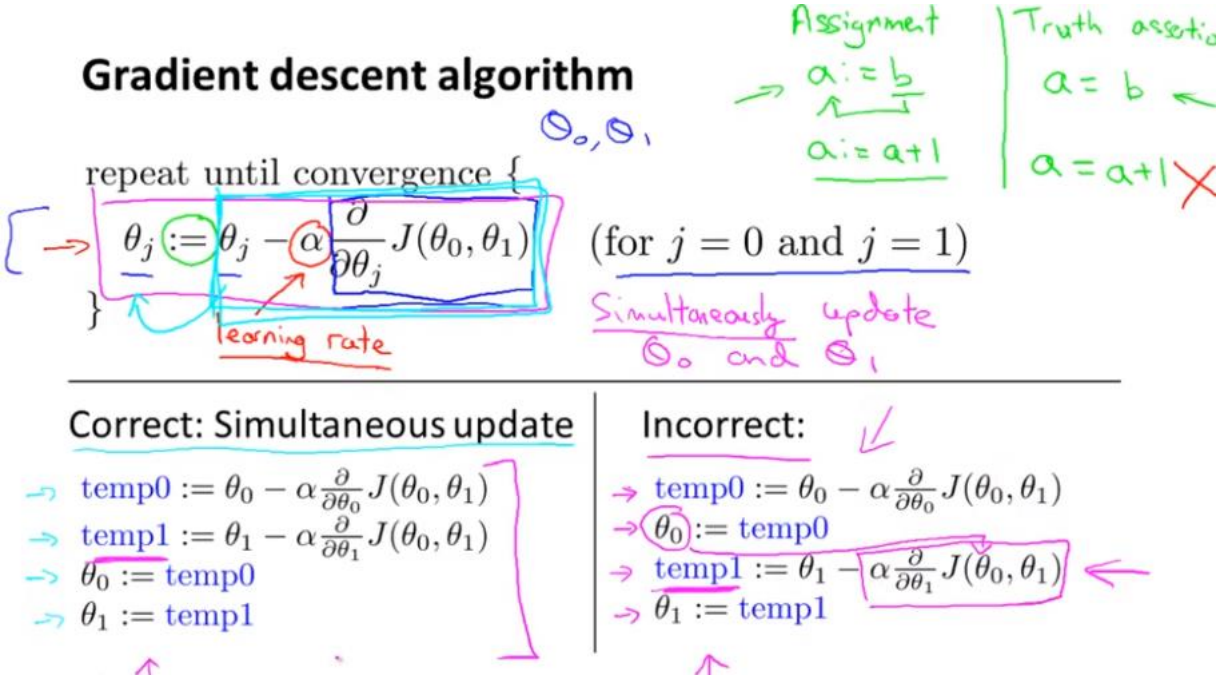
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

:=라는 것은 assignment을 의미하는 것이다.

=라는 것은 truth assertion을 의미한다.

α : learning rate → 얼마나 이동하는가. α가 크면 babystep이 굉장히 큰 것이다.

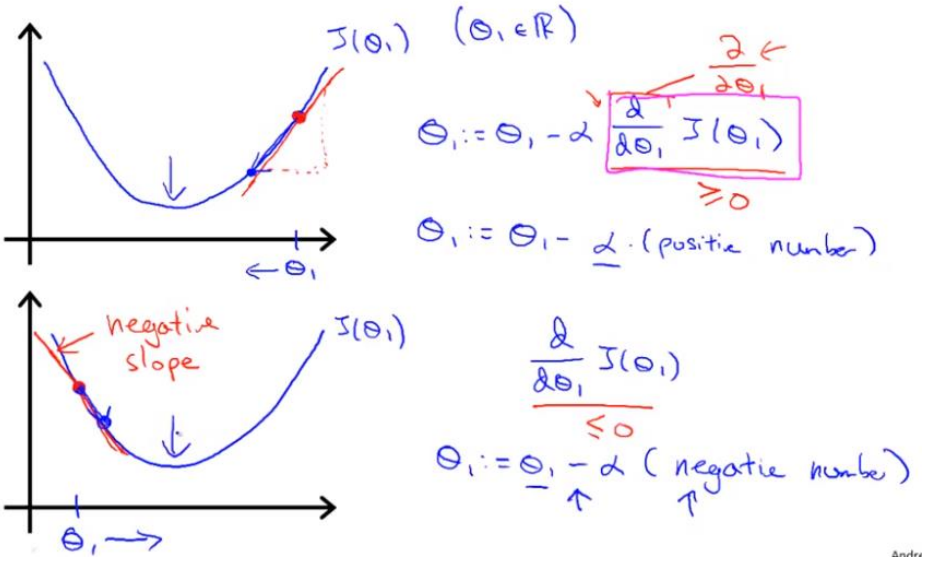
중요 포인트(for j=0 and j=1) : simultaneous update이다. θ_0, θ_1이 동시에 바뀌어야 한다. 만약에 non-simultaneous update는 gradient descent라고 하지는 않는다. 그것은 잘 작동할 수도 있지만, gradient descent라고 부르지는 않고 다른 파생 알고리즘이다.



partial derivative term : 기울기를 의미한다. (방향)

- 이 알고리즘에 대한 직관 :

- Derivative term : J(θ1)의 경우를 생각해보자.

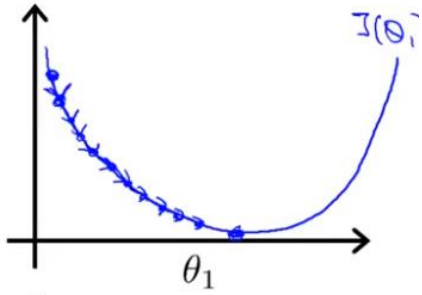


→ 가운데로 θ가 이동하고 있다.

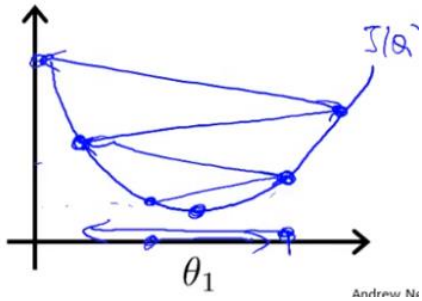
- α (learning rate)가 너무 크면 어떻게 될까?

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

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.



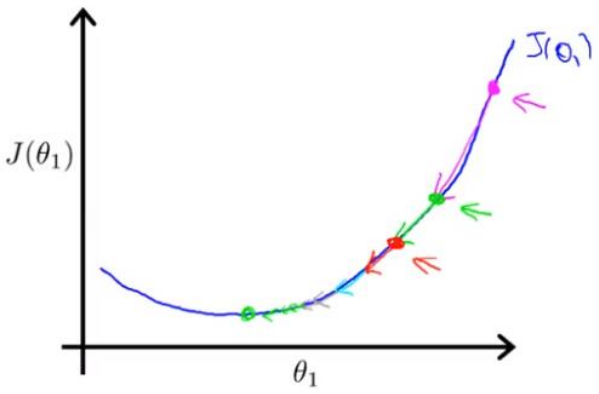
- 만약에 local minimum에 있게 되면 1 step of gradient descent로 인해서 θ가 변하지 않는다. 미분값이 0이 되기 때문이다.

- 1 step of gradient descent의 특징 : local minimum에 가까워질수록 작은 step을 가게 된다. (α가 불변이더라도)

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.



1.2.3. Coding Gradient descent for linear regression (실제 코딩)

- linear regression model에 어떻게 gradient descent를 적용시킬 수 있을까?
- "Batch" gradient descent
  - Batch means each step of gradient descent uses all the training examples. 다 사용!.

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

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

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

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

- 미분을 계산해보자.

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

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

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

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

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y)\, x_j
\end{aligned}
$$

Gradient descent algorithm

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

→ x0 =1로 설정하면 vectorization해서 한 줄로 표현할 수 있다.

h (x) = θ0 * x0 + θ1 * x1 ➔ θ * X

Gradient Descent

Previously (n=1):

Repeat {

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

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

(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 $j = 0, \ldots, n$)

}

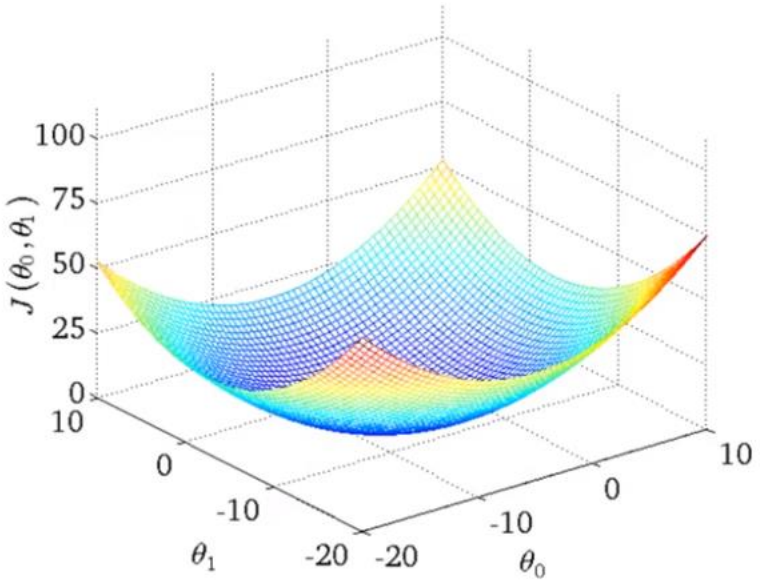
$x_0^{(i)} = 1$

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

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

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

- linear regression의 cost function은 convex function으로 bowl shape이다. 따라서 무조건 local minimum이 global minimum이 된다.

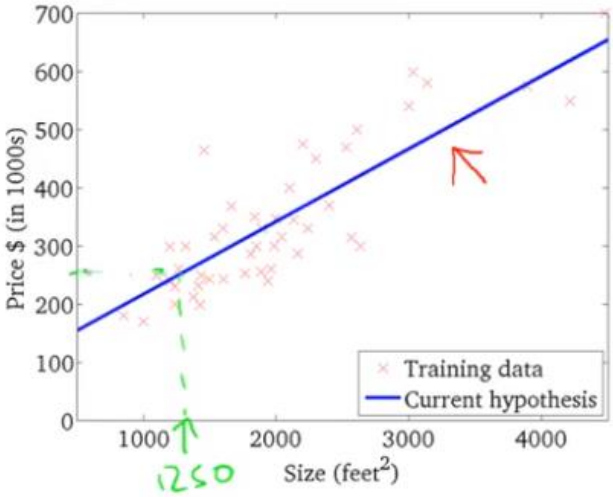


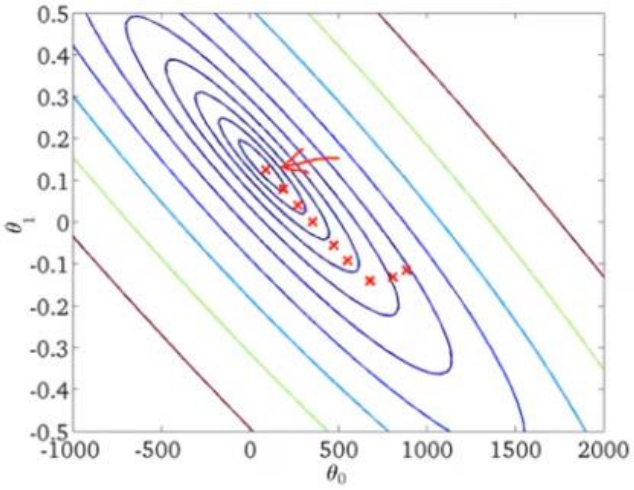
$h_\theta(x)$

(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)





- 부가 설명 : linear regression을 할 때 gradient descent과 같은 iterative algorithm말고 선형대수학에서 배운 최소화 방법이 있다. "Normal equation method"가 그것이다. 그러나 larger data sets에서 전자의 효과가 더 좋은 것으로 밝혀졌다.

1.2.4. Gradient descent in Multivariate linear regression (Week 2)

➔ Gradient descent for multiple variables

- 여러 개의 feature(특징)을 통해서 집값을 예측하고 싶을 때의 선형 회귀

Size (feet²)	Price ($1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

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

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
...	...	...	...	...

$m = 47$

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

Notation:

- n = number of features ($n = 4$)
- $x^{(i)}$ = input (features) of i^{th} training example.
- $x_j^{(i)}$ = value of feature j in i^{th} training example.

→ univariate과 다르게 multivariate은 여러 변수로 hypothesis를 만든다.

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

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

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$[\theta_0\ \theta_1 \cdots \theta_n] = \theta^T$

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n \quad (x_0 = 1)$$

$$= \theta^T x.$$

→ x_0이 1이라고 가정하면 된다! 이를 bias unit이라고도 한다. (neural network에서 사용하는 용어)

→ 내적, 즉 dot product로 표현할 수 있다.

- 이전과 같으나, x와 θ를 n+1차원의 벡터로 생각하면 된다.

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

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$ (θ) n+1 - dimensional vector

Cost function:

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

$J(\theta)$

Gradient descent:

Repeat {

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

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

- cost function 코딩 방법

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

For linear regression, which of the following are also equivalent and correct definitions of $$J(\theta)$$?

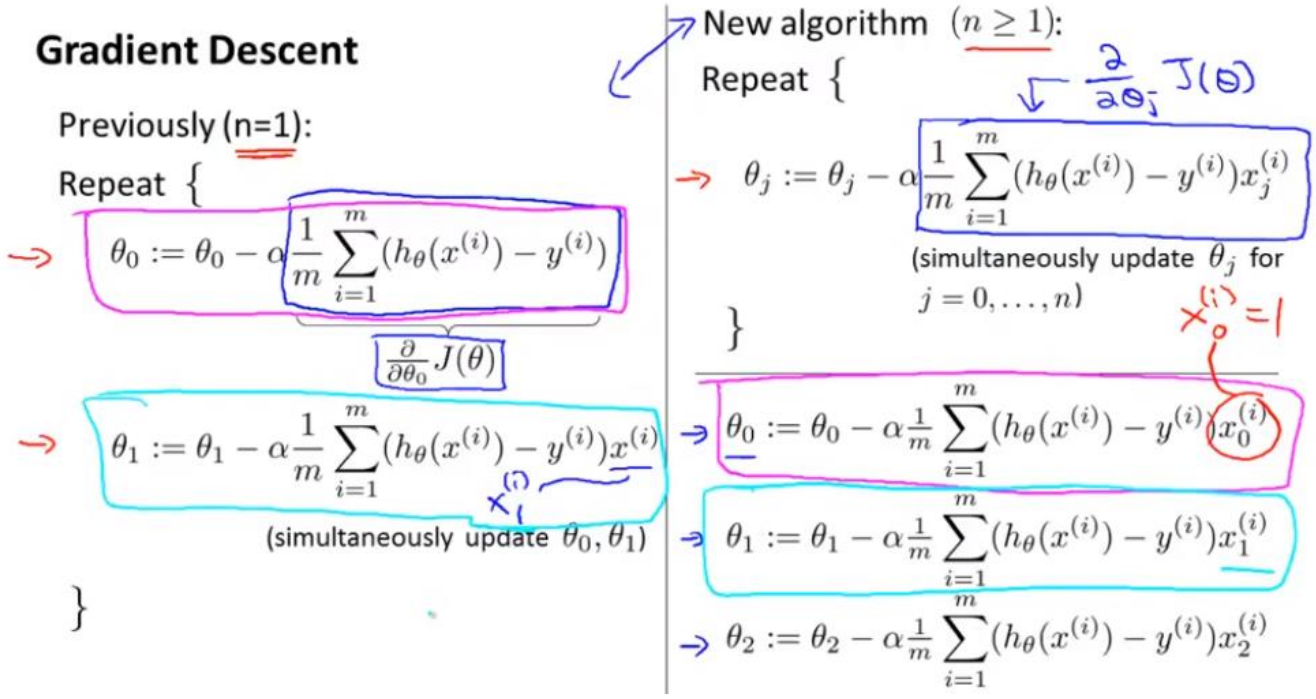
- [x] $$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2$$

  Correct

- [ ] $$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=0}^{n}\theta_j x^{(i)}_j\right) - y^{(i)}\right)^2$$ (Inner sum starts at 0)

  This should be selected

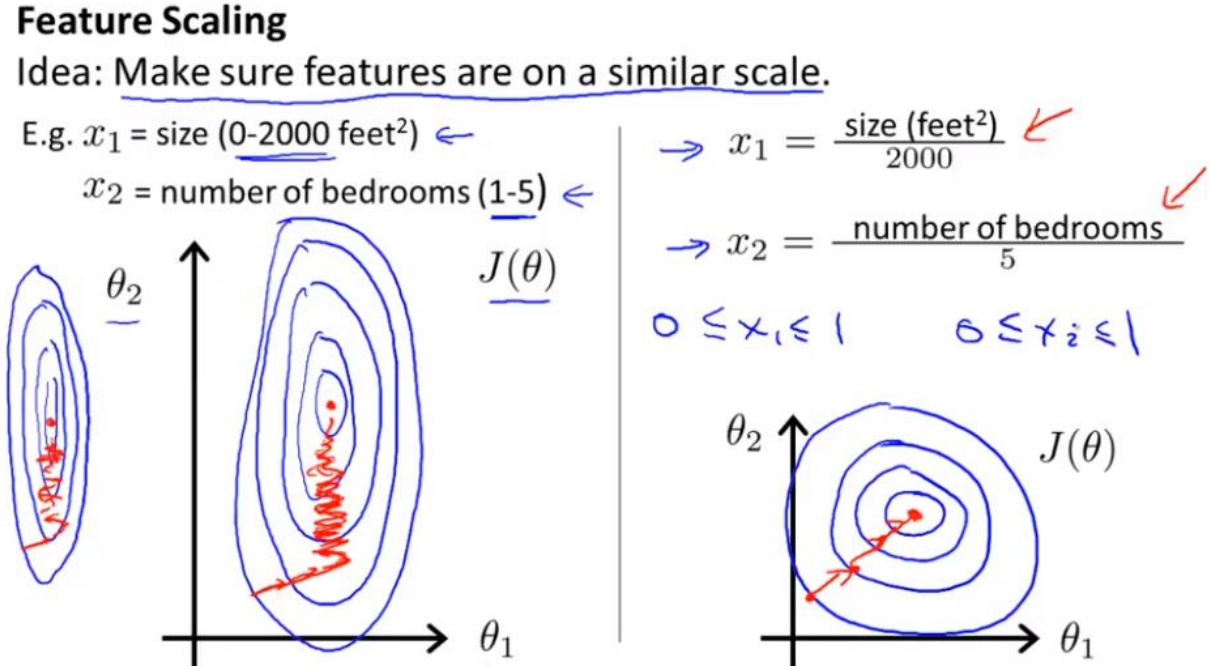
- update rule : 미분값을 계산해 넣으면 X_0가 1로 놓았기 때문에 일반화가 가능하다. 이를 bias unit이라고도 한다.(neural network에서 사용하는 용어)



1.2.5. Gradient descent in practice-1 : feature scaling

- 변수의 단위가 같아야 잘 된다. 너무 길쭉하면 gradient descent가 너무 오래 걸린다. ➔ similar ranges of values / 균일화해서 어디서 시작해도 무방하게 하려고 하는 것일 수도 있다.(혁동님) / 만약에 x0=0, 1000, 2000이라면 α*1/m*합*x0이 들쑥날쑥 될 테니 그것을 방지하려고 하는 것 아닐까?(기석)

(방법 1) feature scaling을 하면 된다. -1<xi<1 range가 되면 좋다. 0<x<3도 좋다. -2<x<1도 좋다. -0.0001<x<0.0001은 안 좋다. 결론적으로 -3 < x < 3이면 괜찮다.



(방법 2) mean normalization도 좋다.(표준화!!)

- X_0에는 하지 말 것
- 나눌 때, range로 해도 되고(max-min), standard deviation으로 해도 된다. 엄청 엄밀할 필요는 없다.

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

E.g. $x_1 = \frac{size - 1000}{2000}$ Average size = 1000

$x_2 = \frac{\#bedrooms - 2}{5}$ 1-5 bedrooms

$-0.5 \le x_1 \le 0.5$, $-0.5 \le x_2 \le 0.5$

$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ (μ_1: avg value of x_1 in training set; s_1: range (max - min) (or standard deviation))

$x_2 \leftarrow \frac{x_2 - \mu_1}{s_2}$

- 코딩

$$x_i := \dfrac{x_i - \mu_i}{s_i}$$

μ_i is the average of all the values for feature (i)

s_i is the range of values (max - min), or s_i is the standard deviation

1.2.6. Gradient descent in practice-2 : learning rate (α) and "Debugging"

- 목표 :

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

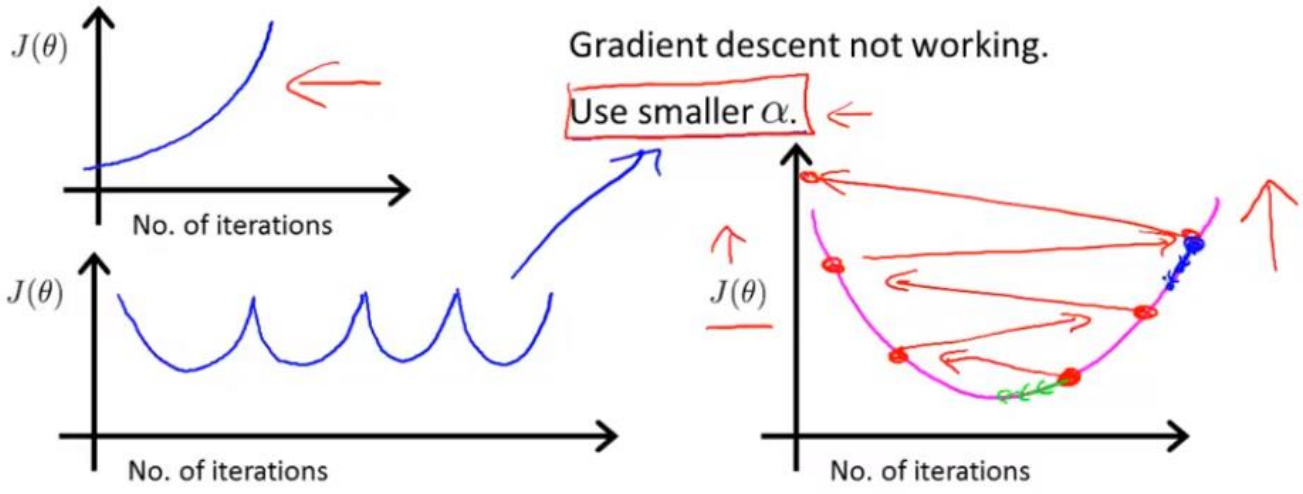
- "Debugging": How to make sure gradient descent is working correctly.
- How to choose learning rate α.

- Make sure gradient descent is working correctly

  : J(θ) should decrease after every iteration → it converges

  cf) automatic convergence test도 있다.

Making sure gradient descent is working correctly.



- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

→ 이렇게 J - iteration 횟수 그래프를 그리면 잘 되는 것이다.

- 잘 안 되는 경우 :

(1) 만약 수렴하지 않을 때에는 α (learning rate)를 줄인다.

(2) 파도 칠 때에도 마찬가지 솔루션을 가진다.

(3) 수렴이 너무 천천히 되면 α를 증가시킨다 → 증가의 속도는 3배, 10배 정도가 괜찮다.

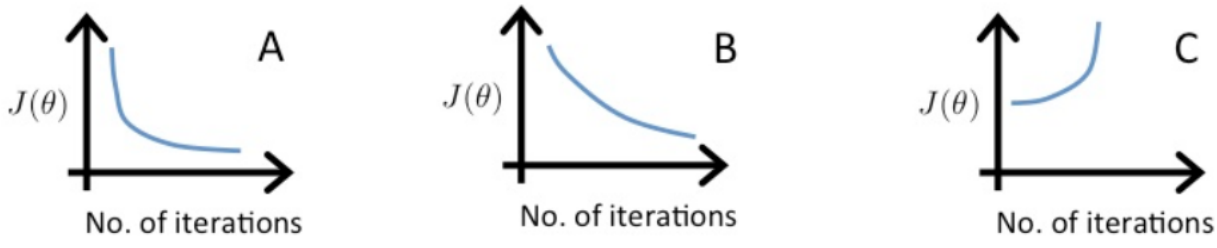
To choose α, try

$$\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$$

3x ≈3x 3x ≈3x

- 문제의 예 :

Suppose a friend ran gradient descent three times, with $$\alpha = 0.01$$, $$\alpha = 0.1$$, and $$\alpha = 1$$, and got the following three plots (labeled A, B, and C):



Which plots corresponds to which values of $$\alpha$$?

- A is $$\alpha=0.01$$, B is $$\alpha = 0.1$$, C is $$\alpha =1$$.
- A is $$\alpha=0.1$$, B is $$\alpha = 0.01$$, C is $$\alpha =1$$.

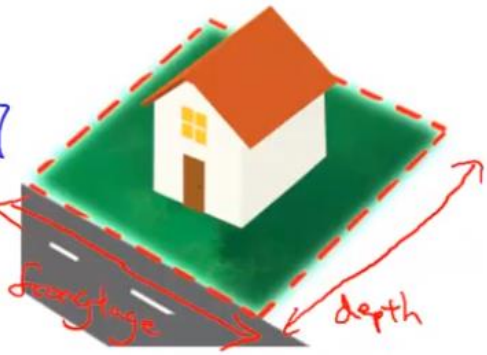
1.2.7. Gradient descent in practice-3 : Choosing features and polynomial regression

(1) 두 개를 쓰는 것보다 곱한 한 개의 feature를 쓰는 게 나을 수 있다.

Housing prices prediction

$$h_\theta(x) = \theta_0 + \theta_1 \times frontage + \theta_2 \times depth$$

x_1 x_2



Area

x = frontage * depth

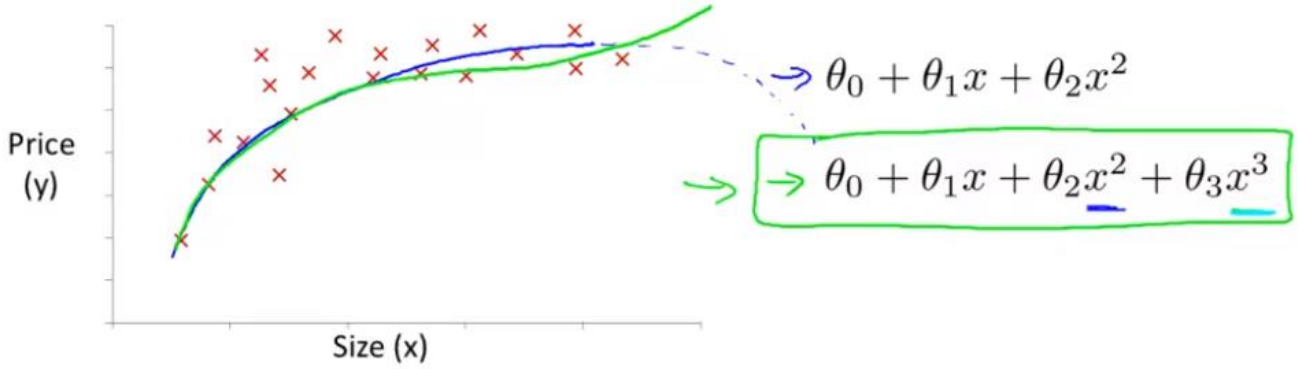
$h_\theta(x) = \theta_0 + \theta_1 x$

↑ land area

(2) Polynomial regression : quadratic model

- 방법 : 제곱, 세제곱한 것을 feature로 정하면 된다.

Polynomial regression

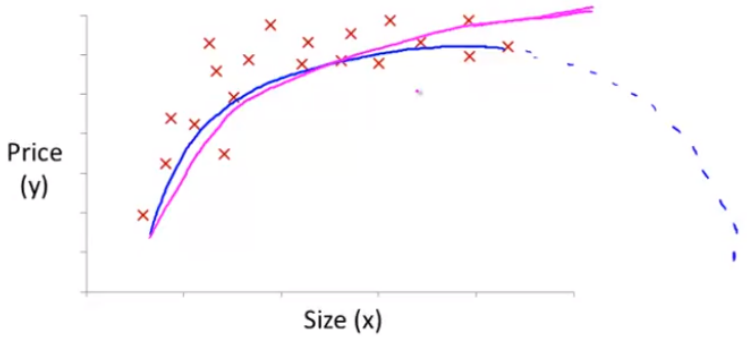


$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

- 주의 : scaling을 꼭 한다. range로 나누면 된다.
- cubic model로 안 가도록 하고 싶을 때! square root를 사용한다!!! (2차 함수는 밑으로 포물선을 그리기 때문이다.

Choice of features



$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

1.2.8. Gradient descent in practice-4 : Regularized Gradient descent

- 문제 정의 : polynomial 혹은 many feature를 쓰게 되면 overfitting 문제가 일어날 수 있다.
- 해결 : 이를 해결하기 위해서 regularize시키자. θ를 줄이는 cost function 만들기.

Gradient descent

Repeat {

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \ldots, n)$$

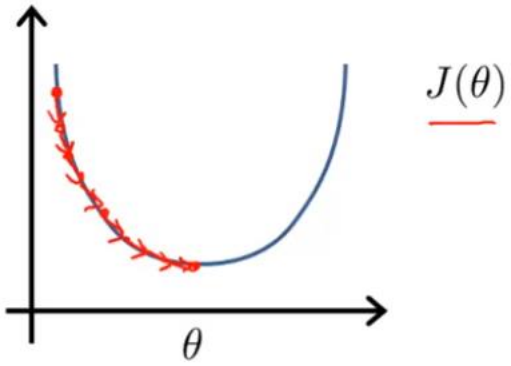
}

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

→ J cost function을 편미분하면 λ/m * θ가 나온다. θ^2를 미분하면 2가 내려오기 때문이다. 따라서 괄호 안에 (1-α*λ/m)이라고 쓸 수 있다. (1-α*λ/m) < 1이다. 거의 0.99.

1.3. Normal equation ← Supervised machine learning - regression - solution2

Gradient Descent



Normal equation: Method to solve for θ analytically.

→ iterative하지 않게 바로 갈 수 있다.

1.3.1. 방법

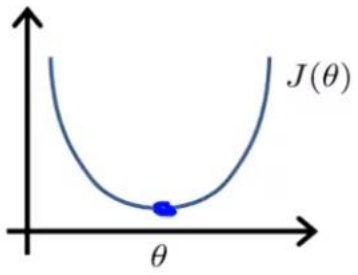
- 직관 : 미분해서 0으로 하면 된다.

Intuition: If 1D ($\theta \in \mathbb{R}$)

→ $J(\theta) = a\theta^2 + b\theta + c$

$\frac{d}{d\theta} J(\theta) = \cdots \overset{\text{set}}{=} 0$

Solve for θ



$\theta \in \mathbb{R}^{n+1}$ $\quad J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots \overset{\text{set}}{=} 0$ (for every j)

Solve for $\theta_0, \theta_1, \ldots, \theta_n$

And

- 예시 :

(1) 먼저 data set로 행렬을 만든다. 행렬 x와 벡터 y로 만든다.

Examples: $m = 4$.

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad m \times (n+1)$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \quad m\text{-dimensional vector}$$

- 행렬 X를 구하는 방법

m **examples** $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$ **;** n **features.**

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\text{(design matrix)} \times = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \quad m \times (n+1)$$

→ m이 데이터의 개수이고, n은 변수의 개수이다.

(2) θ를 구하면 끝난다.

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

- octave에서

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

$(X^T X)^{-1}$ is inverse of matrix $X^T X$.

Set $A = X^T X$

$(X^T X)^{-1} = A^{-1}$

Octave: `pinv(X'*X)*X'*y`

`pinv(X'*X)`

- 지금 이 normal equation method에서는 feature scaling을 안 해도 된다.

왜냐하면 표준화의 과정이 포함되어 있기 때문이다.

<괴장히 중요!!!> 왜 $X * \theta = y$ 가 왜 최소화하는 해를 갖는 것인지!!!

- $X\theta = y$

$\theta = X^{-1} * y$

이렇게 하면 되지만 역행렬이 없는 경우가 있을 수 있잖아!

- $X\theta = y$

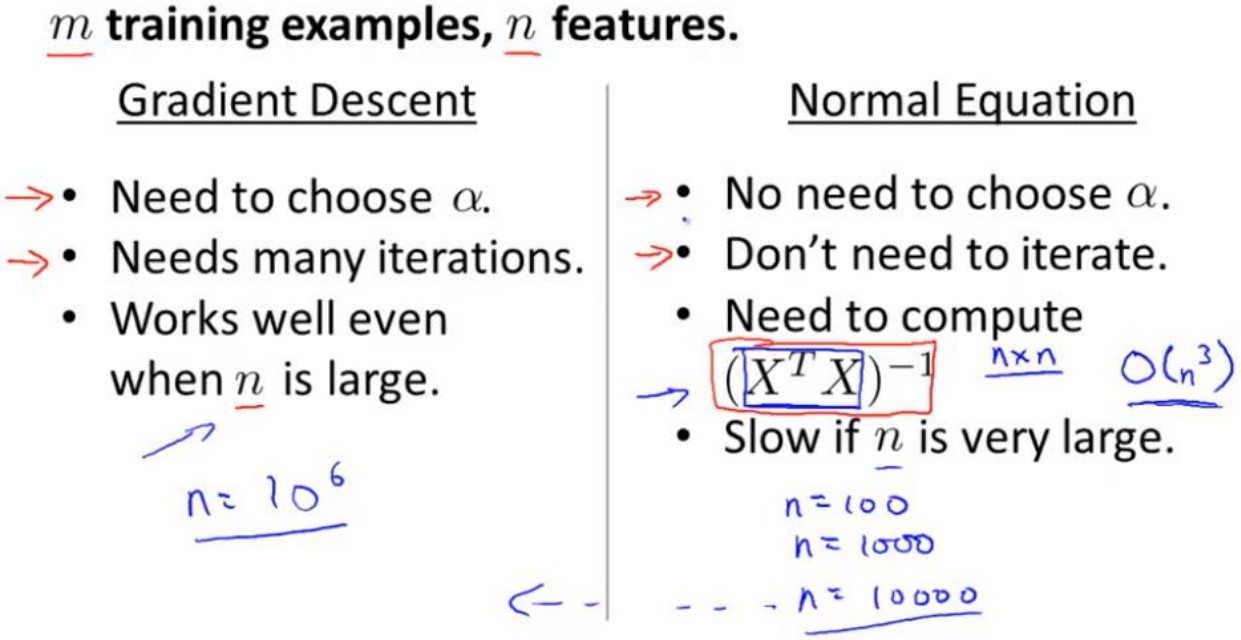
$X^T * X * \theta = X^T * y$

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

→ $X * \theta = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_3 + \theta_4x_4 + ... \approx y$ (y랑 완전히 같아지지는 않는다.!!! 그래도!!!)

1.3.2. 장/단점

- n이 크면 gradient descent를 하면 된다. (n=10,000정도 이상이면 크지.. 역행렬 계산이 너무 expensive)
- n이 1000보다 작으면 normal descent가 좋다.
- classification 알고리즘은 normal equation이 작동 안 된다. (정말??)



1.3.3. Normal equation non-invertibility

- 만약에 X^TX is non-invertible(singular / degenerate)하면 어떻게 하냐?

→ Octave에서는 pinv(X'*X)*X'*y 해도 잘 된다. (pseudo inverse) → 저번 시간에 했던 것이다! 역행렬이 없는 경우에 바로 역행렬을 구하지

- invertible이 안 되는 경우! $X^T * X$가 역행렬이 없는 경우

(1) linearly dependent한 경우 : redundant하면 역행렬이 없다.

→ 다중 공선성 문제 : 똑같은 변수가 여러 개 집어넣으면!! → 비슷한 변수를 빼야 한다. 교차상관성을 다 계산해야 한다. 괴장히 중요하다!!

(2) feature가 너무 많은 경우 : 너무 feature(n)가 많으면 안 될 수도... 해결하려면 regularization이 필요하다. 나중에 설명???? 바로 뒤에 나옴. 1.3.4.

→ feature가 sample보다 더 많은 경우, degree of freedom 개념 : 샘플 개수보다 특징 수가 많으면 특징들의 상관관계를 알 수가 없다. 부족하다.

What if $X^T X$ is non-invertible?

- Redundant features (linearly dependent).
  E.g. x_1 = size in feet^2
  x_2 = size in m^2

  $1_m = 3.28$ feet

  $x_1 = (3.28)^2 x_2$

  $\rightarrow n = 10$
  $\rightarrow n = 100$
- Too many features (e.g. $m \leq n$). $\theta \in \mathbb{R}^{101}$
  - Delete some features, or use regularization.

cf)

Cross-covariance

$$\gamma_{XY}(\tau) = \mathrm{E}[(X_t - \mu_X)(Y_{t+\tau} - \mu_Y)],$$

Cross correlation coefficient

$$\rho_{XY}(\tau) = \frac{1}{\sigma_X \sigma_Y} \mathrm{E}[(X_t - \mu_X)(Y_{t+\tau} - \mu_Y)] = \frac{1}{\sigma_X \sigma_Y} \gamma_{XY}(\tau)$$

1.3.4. Normal equation in practice : regularized normal equation

Normal equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \quad m \times (n+1)$$

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \mathbb{R}^m$$

$\rightarrow \min_\theta J(\theta)$

$\frac{\partial}{\partial \theta_j} J(\theta) \overset{\text{set}}{=} 0$ for all j

$\rightarrow \theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$

→ derive가 가능하다.

- 일석이조!! : regularization을 하면 non-invertible한 XTX도 invertible해진다. (증명 가능)

Non-invertibility (optional/advanced).

Suppose $m \leq n$,
(#examples) (#features)

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

non-invertible / singular

pinv inv

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

invertible.

→ m < n하면 해가 정해지지 않기 때문에 non-invertible하다. positive definite matrix (Determinant이 무조건 0보다 커진다)

Supervised learning 종류 : regression problem / classification problem

Week 3.

1.4. regression problem : 위 참조

1.5. Classification

1.5. Classification ➔ logistic regression

1. Why logistic regression?

- 목적 : logistic regression을 위해서!! discrete value를 가졌을 경우! continuous가 아닌 경우
- 종류 : two class, multiclass classification

Classification

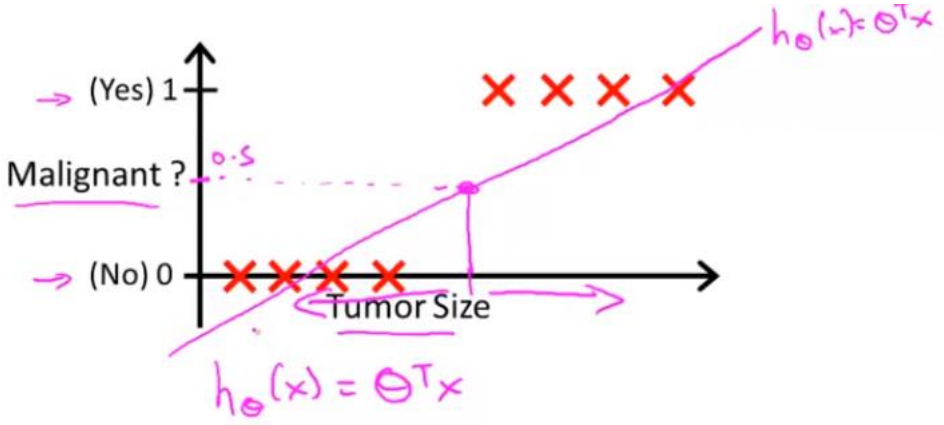
→ Email: Spam / Not Spam?
→ Online Transactions: Fraudulent (Yes / No)?
→ Tumor: Malignant / Benign ?

→ $y \in \{0, 1\}$
0: "Negative Class" (e.g., benign tumor)
1: "Positive Class" (e.g., malignant tumor)

→ $y \in \{0, 1, 2, 3\}$

- 기존 linear regression의 한계 : linear regression으로 classification을 하는 경우의 문제점

(1) 이상값이 존재하게 되면 맞지 않는 hypothesis, threshold를 갖게 되는 문제점

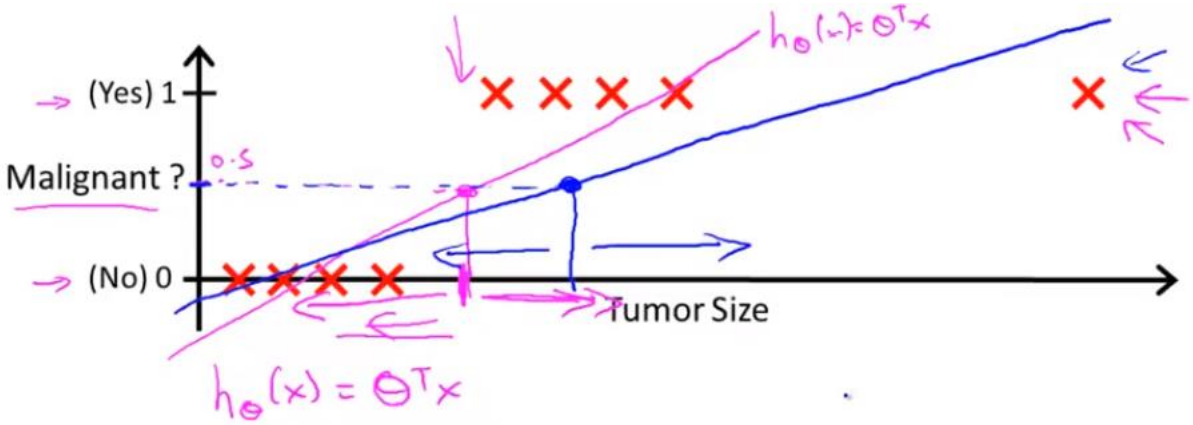


→ Threshold classifier output $h_\theta(x)$ at 0.5:

→ If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

→ 그런데 아웃라이어가 있게 되면, linear regression이 부정확한 threshold를 갖게 한다.



→ Threshold classifier output $h_\theta(x)$ at 0.5:

→ If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

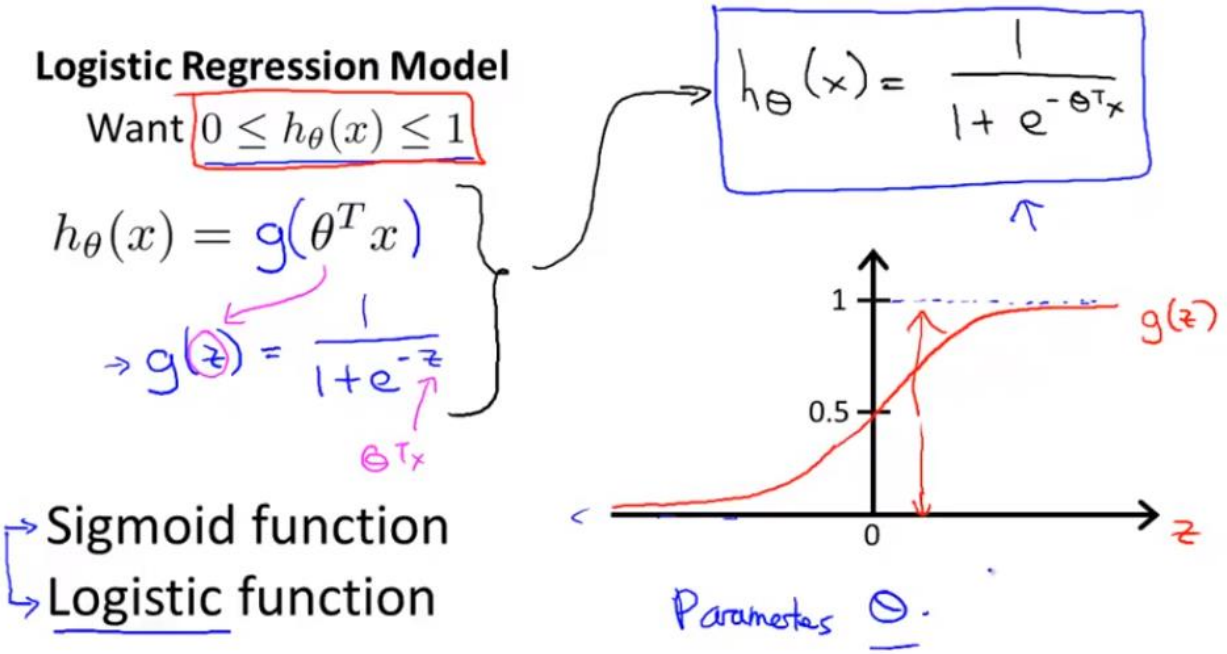
(2) hypothesis $h_\theta(x)$가 1보다 크거나, 0보다 작은 경우가 생긴다. 하지만 classification에서는 y=0, 1이 전부이다.

- 문제점 해결 : **logistic regression**을 하게 되면 0 <= $h_\theta(x)$ <= 1이 된다. (regression이라고 칭하기는 하지만, classification model이니까 헷갈려 하지 말 것.)

2. What is Logistic regression model

(1) 수식 : $h\theta(x) = 1 / 1 + e^{-\theta^T x}$

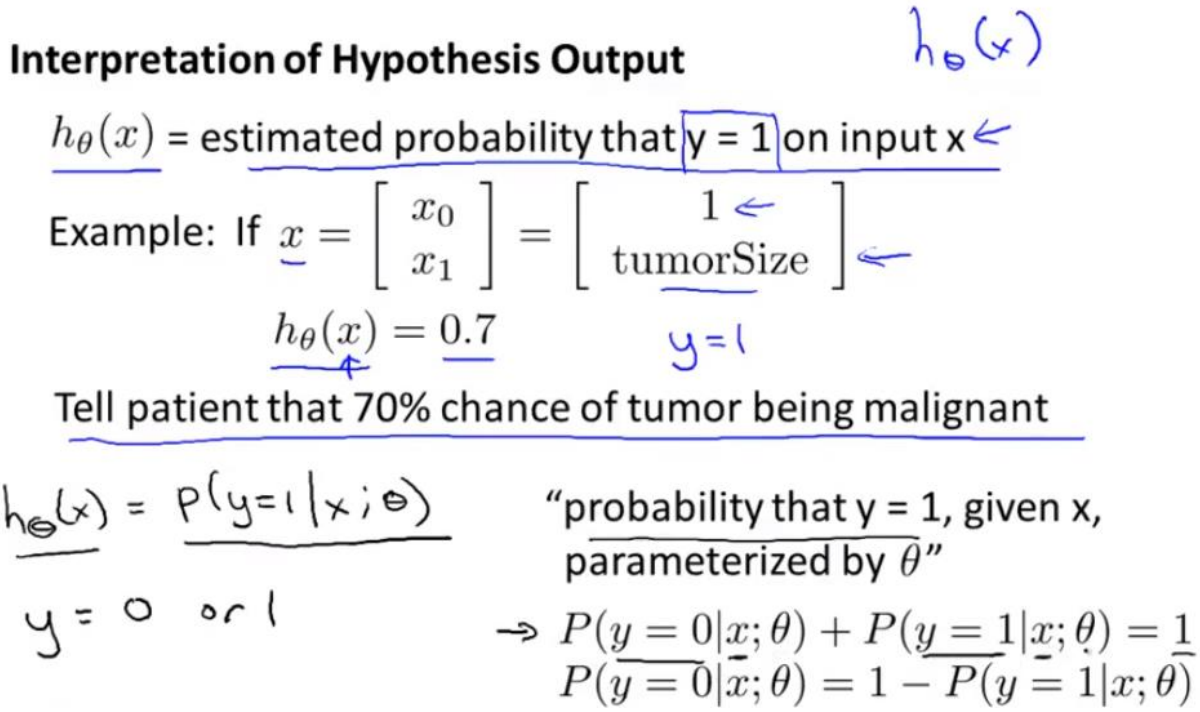
→ 위 아래로 0, 1에 수렴하기 때문에 0와 1을 벗어날 리 없다.



(2) 의미 : hθ(x) = estimated probability that y=1 on input x

ex) 환자 1명의 tumor size x가 주어졌을 때, 악성 종양(y=1)일 가능성이 몇 퍼센트이냐?

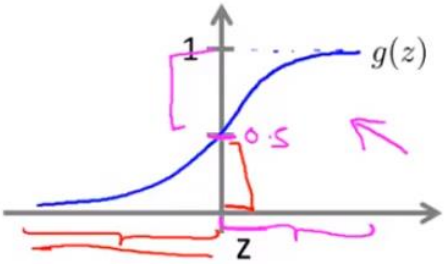
h θ (x) = P (y=1 | x ; θ) ← probability that y=1, given x, parameterized by θ



(3) Decision boundary

- 0.5가 기준이 될 수 있다. h θ (x) >= 0.5 이면 y=1로 predict하고, h θ (x) < 0.5이면 y=0로 predict하면 된다. 다시 말하면, h θ (x) = g (z) = g (θ ᵀ x) = 0.5가 되는 θ ᵀ x 가 0이 기준이다. θ ᵀ x가 0보다 크면 y=1로, θ ᵀ x가 0보다 작으면 y=0로 prediction하면 된다.

Logistic regression



$h_\theta(x) = g(\theta^T x) = P(y=1 \mid x; \theta)$

$g(z) = \frac{1}{1+e^{-z}}$

Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

$\theta^T x \geq 0$

$g(z) \geq 0.5$ when $z \geq 0$

$h_\theta(x) = g(\theta^T x) \geq 0.5$ whenever $\theta^T x \geq 0$

predict "$y = 0$" if $h_\theta(x) < 0.5$

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

$\theta^T x < 0$

$g(z) < 0.5$

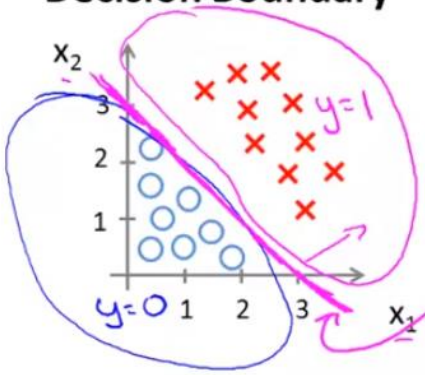
So if our input to g is $\theta^T X$, then that means:

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$when\ \theta^T x \geq 0$$

- 예를 들면, $\theta^T x$ 가 >= 0 이면, 즉, g(z) = h θ (x)가 0.5보다 크면 y=1이라고 추정할 수 있다. 이 경계를 decision boundary라고 하며, 이는 data set에서 파생되는 속성이 아니라, h θ (x), 즉 hypothesis에서 알 수 있는 속성이다.

Decision Boundary

$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

-3 1 1

Decision boundary

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$\theta^T x$

$x_1 + x_2 \geq 3$

x_1, x_2

$h_\theta(x) = 0.5$

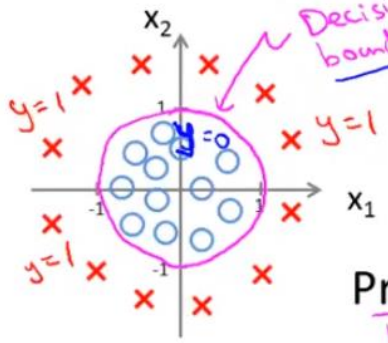
$x_1 + x_2 = 3$

$x_1 + x_2 < 3$

$y = 0$

- non-linear decision boundaries
  - polynomial term을 더하면 이것도 표현할 수 있다.
  - 원으로도 표현 가능

Non-linear decision boundaries



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$x_1^2 + x_2^2 = 1$

$x_1^2 + x_2^2 \geq 1$

3. How to fit the parameter

- 목표 : cost function 을 가장 minimize 시키자!
- 문제 정의 :

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $\quad x_0 = 1, y \in \{0, 1\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters θ ?

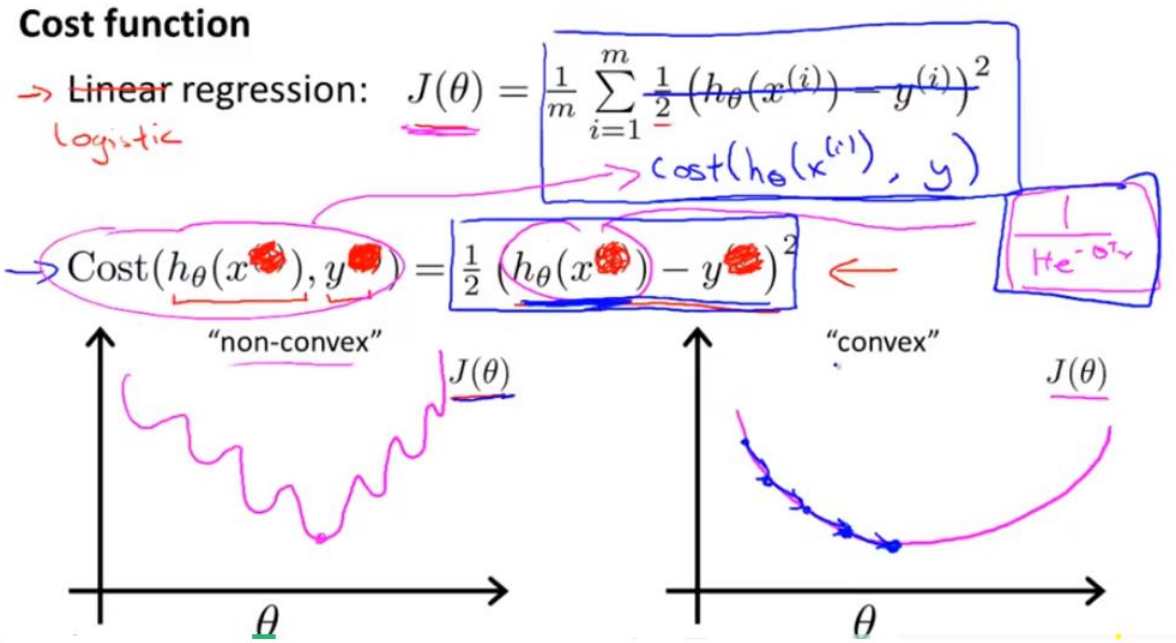
- linear regression 때처럼, h(x) - y 의 제곱의 합으로 할 때의 문제점 → non convex 해지기 때문에 gradient descent 를 하면 하나의 global minimum 에 수렴하지 않는다. 따라서 cost function 이 convex 해지는 방법을 찾아야 한다.

Cost function

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

$\text{cost}(h_\theta(x^{(i)}), y)$

$$\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$



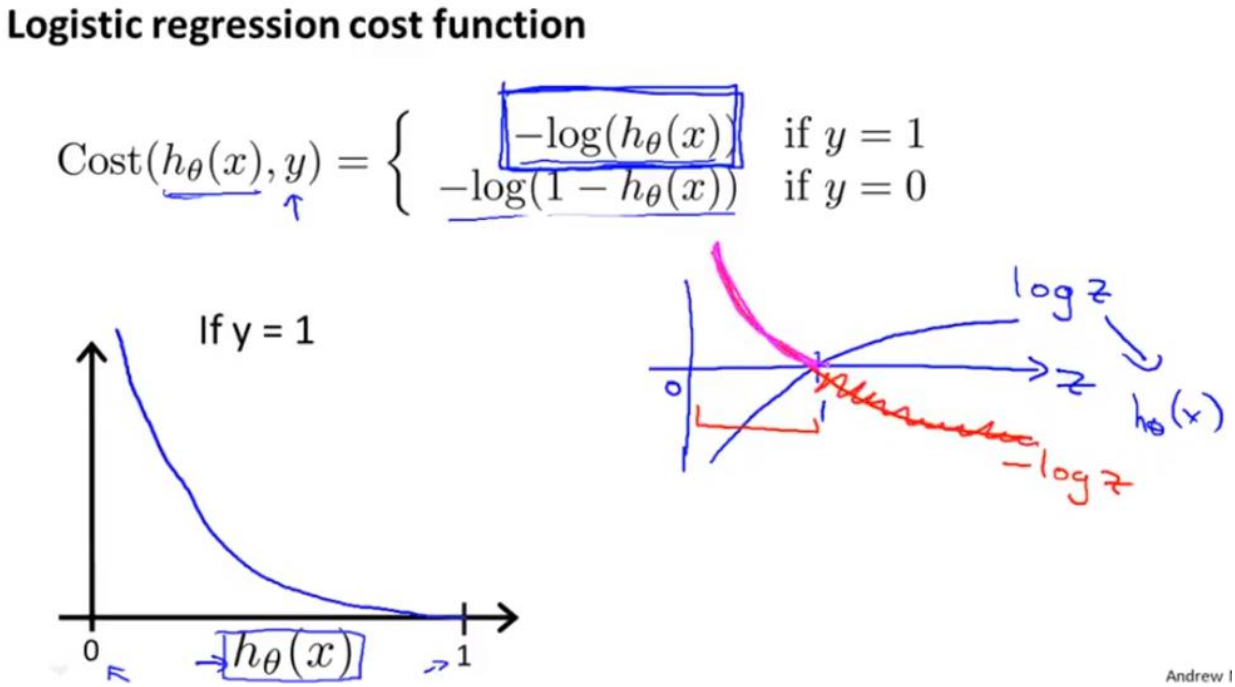
질문!!!

이거 왜 non convex해지냐??

- 해결 : log함수를 사용해서

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$
$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$

(1) y = 1일 때,



→ 직관적인 의미 : y=1임에도 불구하고, h θ (x)라고 대답을 해버리면(tumor가 malignant일 확률이 0이라고) cost가 굉장히 크게 되게끔 설정한 것이다.

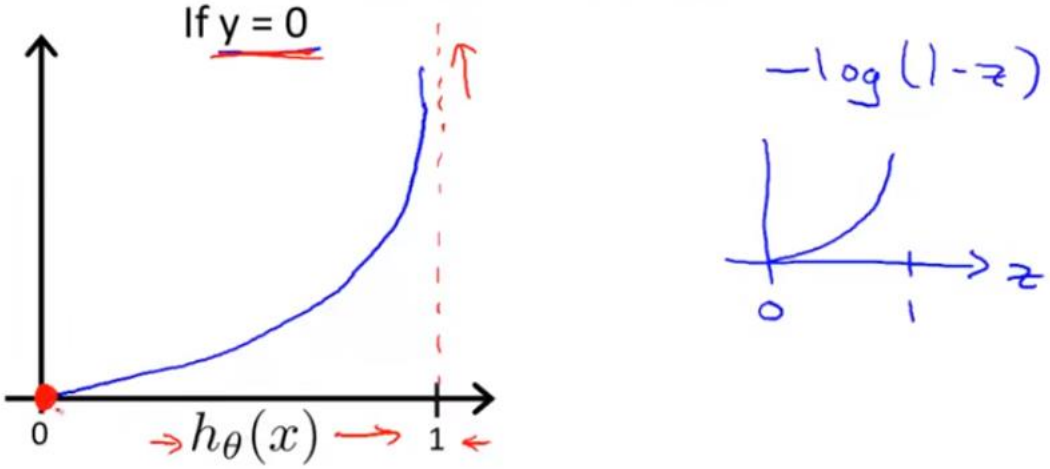
→ $\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

→ Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

(2) y = 0일 때

Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



$\text{Cost}(h_\theta(x), y) = 0$ if $h_\theta(x) = y$
$\text{Cost}(h_\theta(x), y) \to \infty$ if $y = 0$ and $h_\theta(x) \to 1$
$\text{Cost}(h_\theta(x), y) \to \infty$ if $y = 1$ and $h_\theta(x) \to 0$

- 표기 : simplified cost function
→ 두 줄로 쓴 식을 1줄로 쓸 수 있다.

Logistic regression cost function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y) \log(1 - h_\theta(x))$$

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

→ 왜 이것을 사용하냐? maximum likelihood estimation으로도 유도할 수 있다.

To fit parameters θ:

$\min_\theta J(\theta)$ Get θ

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ $p(y=1 \mid x;\theta)$

- fitting 방법 : gradient descent
- 방법 :

이거 어떻게 저렇게 예쁘게 나오는 거지??

Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat {

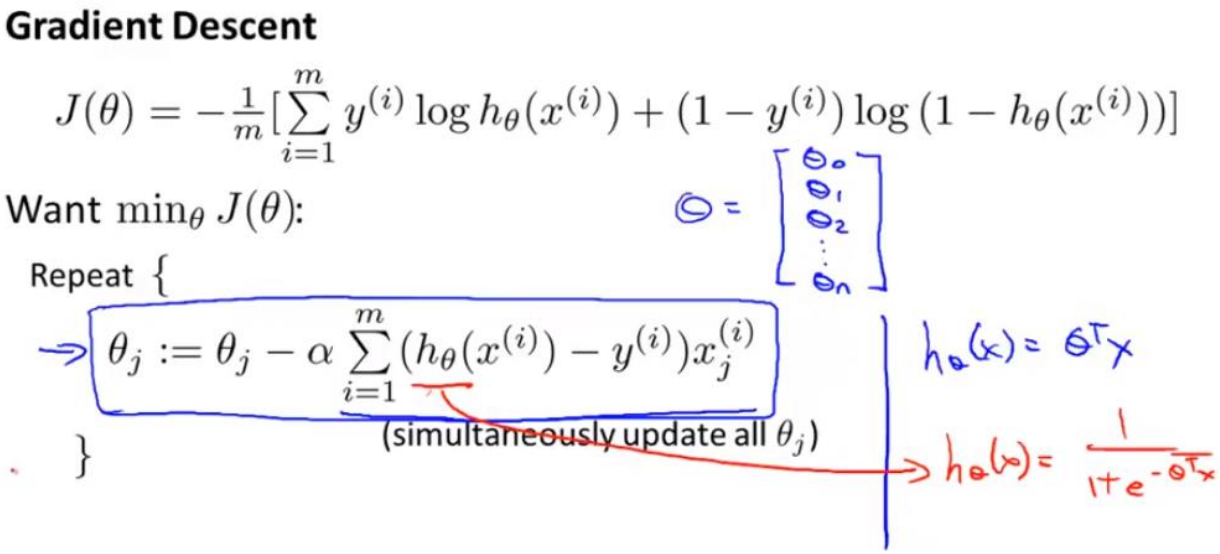
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

(simultaneously update all θ_j)

}

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

→ 유도해봤는데, 매우 비슷하게 나온다. 완전히 성공하지는 못함. 부호가 어딘가 (-)가 들어옴



- α를 맞게 정하는 방법 :

Plot
$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$
as a function of the number of iterations and
make sure $J(\theta)$ is decreasing on every iteration.

→ J가 작아졌다가 커지거나 하면 alpha를 다시 설정해야 한다.

- Vectorization

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

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

$$\vdots$$

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

We would like a vectorized implementation of the form
$\theta := \theta - \alpha\delta$ (for some vector $\delta \in \mathbb{R}^{n+1}$).

What should the vectorized implementation be?

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

- feature scaling도 할 수 있다.

- 나중에 7.2.1.에서 mini-batch gradient descent랑 stochastic gradient descent를 배우게 된다.(computational cost를 줄여주기 위한 것이다.)

4. Advanced Optimization for linear / logistic regression

(1) 방법의 종류

1) gradient descent : 위에서 했다.

Optimization algorithm

Cost function $J(\theta)$. Want $\min_\theta J(\theta)$.

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial\theta_j}J(\theta)$ (for $j = 0, 1, \ldots, n$)

Gradient descent:

Repeat $\{$

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

$\}$

- Advanced methods

2) conjugate gradient

3) BFGS

4) L-BFGS

- 장점 : no need to manually pick α(learning rate)

(2) Octave에서의 방법 :

(2-1) linear regression에서 어떻게 하는지 한번 해보자.

Example: $\min_\theta J(\theta)$

$\theta = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$ $\theta_1=5, \theta_2=5.$

$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

$\frac{\partial}{\partial\theta_1}J(\theta) = 2(\theta_1 - 5)$

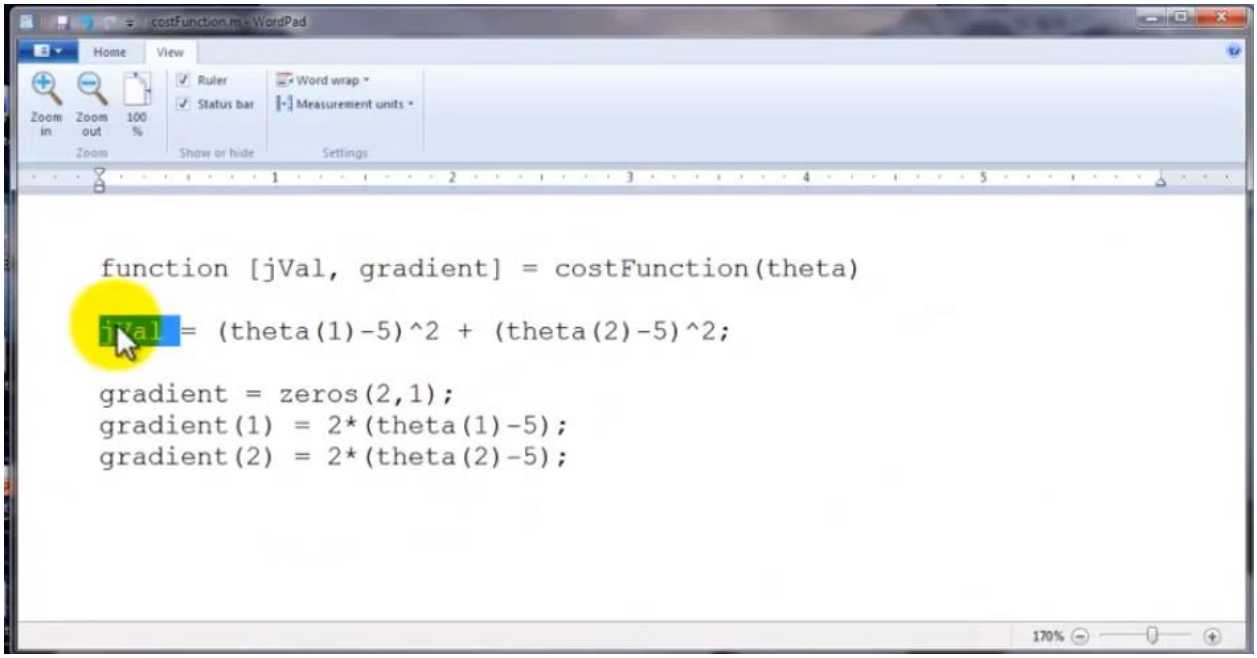
$\frac{\partial}{\partial\theta_2}J(\theta) = 2(\theta_2 - 5)$

```
function [jVal, gradient]
            = costFunction(theta)
  jVal = (theta(1)-5)^2 + ...
            (theta(2)-5)^2;
  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

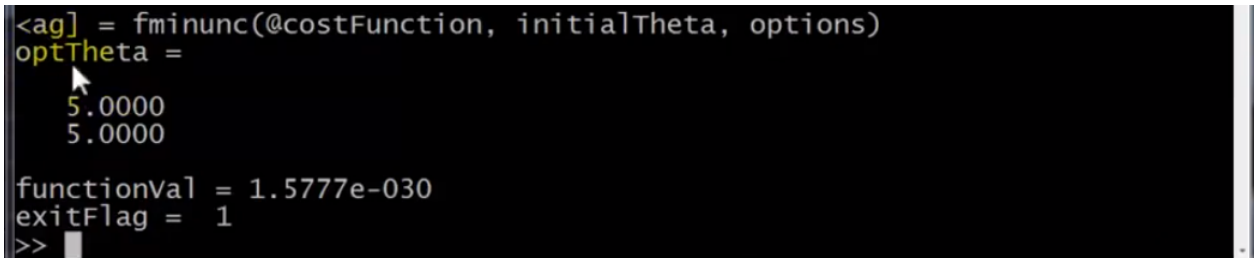
```
options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] ...
      = fminunc(@costFunction, initialTheta, options);
```

→ options = optimset('GradObj', 'on', 'MaxIter', 100) → 100번 iteration을 하라는 거구나.. 이거 혹시 library인가?

- cost function ← 따로 함수 설정

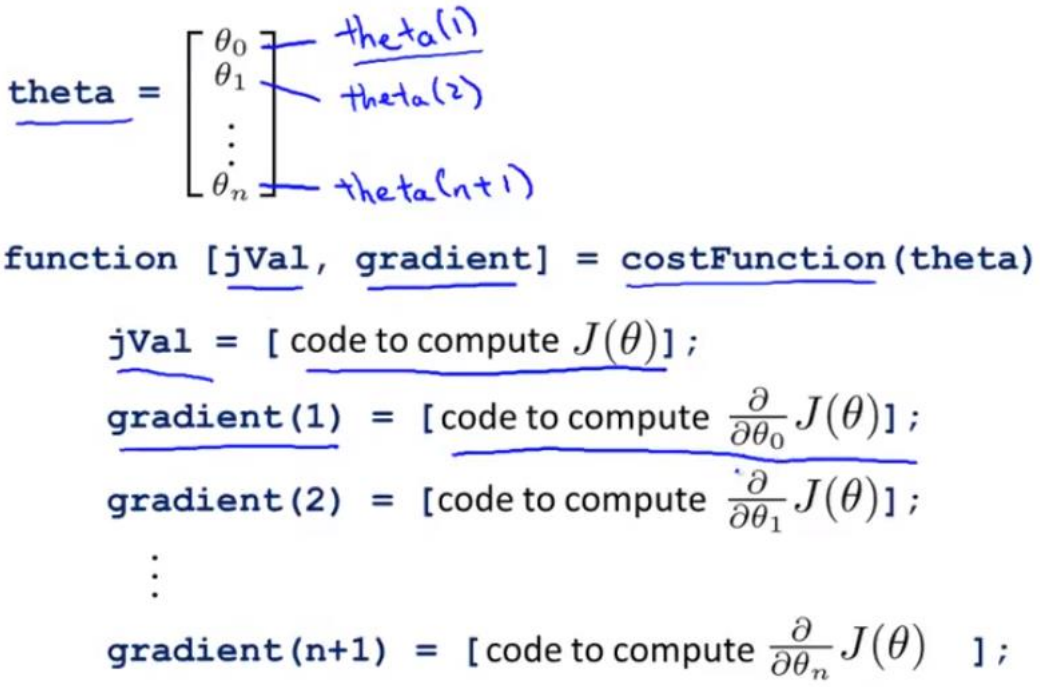


→ 이렇게 치면 결과가 빨리 나온다.



→ Theta가 5, 5가 optimum이구나.

(2-2) logistic regression을 하는 방법 (Gradient descent가 아닌 advanced algorithm)



→ indexing이 1부터 된다는 것을 조심.

- 정리 :

We first need to provide a function that evaluates the following two functions for a given input value θ:

$$J(\theta)$$
$$\frac{\partial}{\partial \theta_j} J(\theta)$$

We can write a single function that returns both of these:

```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

Then we can use octave's "fminunc()" optimization algorithm along with the "optimset()" function that creates an object containing the options we want to send to "fminunc()". (Note: the value for MaxIter should be an integer, not a character string - errata in the video at 7:30)

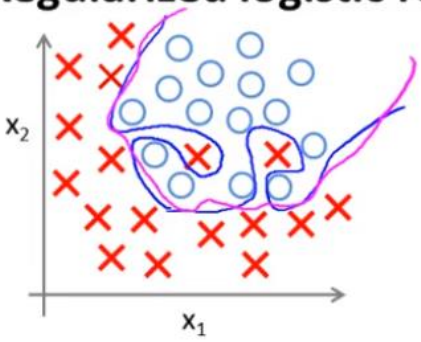
```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta,
     options);

```

We give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.

5. Application of logistic regression-1 : Regularized logistic regression

- 원리 : θ의 합을 도입하여 overfitting 문제를 완화시킨다.

Regularized logistic regression.



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Cost function:

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

$\theta_1, \theta_2, \dots, \theta_n$

- 알고리즘 : hypothesis가 다르지만 외관상 linear regression과 비슷하다. 주의 : j=0을 빼고 하면 된다.

Gradient descent

Repeat {

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

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \dots, n)$$

$\theta_1 \dots \theta_n$

}

$\frac{\partial}{\partial \theta_j} J(\theta)$

$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

- regularized gradient descent가 잘 되는지 확인하는 방법

> Plot $-[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$ as a function of the number of iterations and make sure it's decreasing.
>
> **Correct**

→ j cost function이 감소하는지 봐야지.

- Advanced optimization에서 해보기

Advanced optimization

fminunc (@costFunction, initialTheta, options)

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ theta(1), theta(2), theta(n+1)

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
        J(θ) = [ -(1/m) Σ_{i=1}^{m} y^(i) log(h_θ(x^(i))) + (1 - y^(i)) log 1 - h_θ(x^(i)) ] + (λ/2m) Σ_{j=1}^{n} θ_j^2
    gradient(1) = [code to compute ∂/∂θ_0 J(θ)];
    gradient(2) = [code to compute ∂/∂θ_1 J(θ)];
    gradient(3) = [code to compute ∂/∂θ_2 J(θ)];
      :
    gradient(n+1) = [code to compute ∂/∂θ_n J(θ)];
```

→ fminunc (@costFunction) 하면 j cost function이 minimize된다.

f minimization unconstrained

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta,
     options);
```

We give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.

6. Application of logistic regression-2 : Multiclass classification in logistic regression

- 예시 :

Multiclass classification

Email foldering/tagging: Work, Friends, Family, Hobby

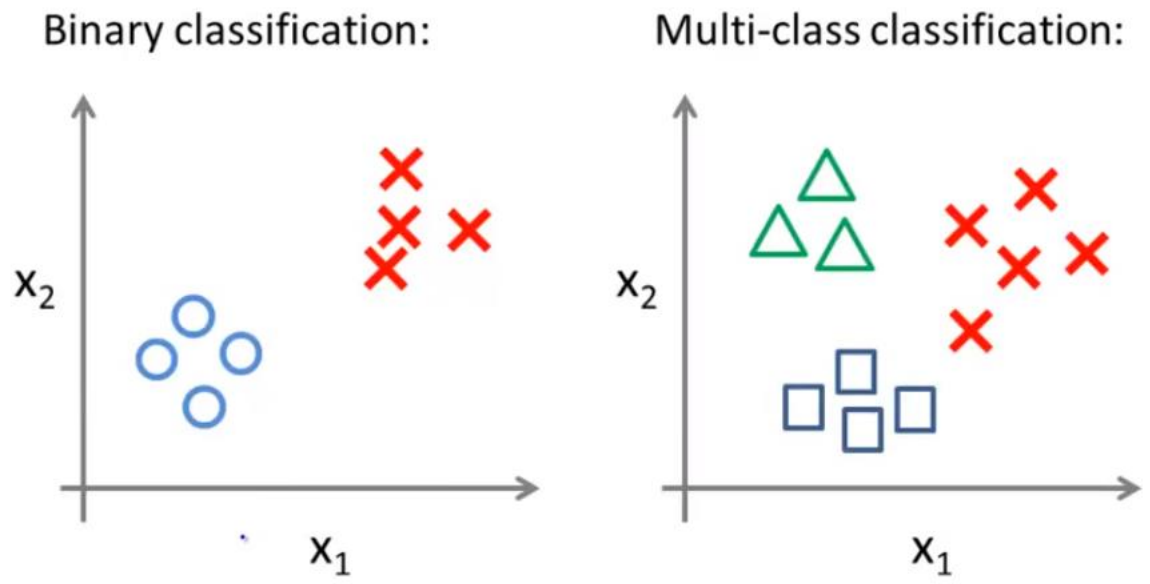
y=1 y=2 y=3 y=4

Medical diagrams: Not ill, Cold, Flu

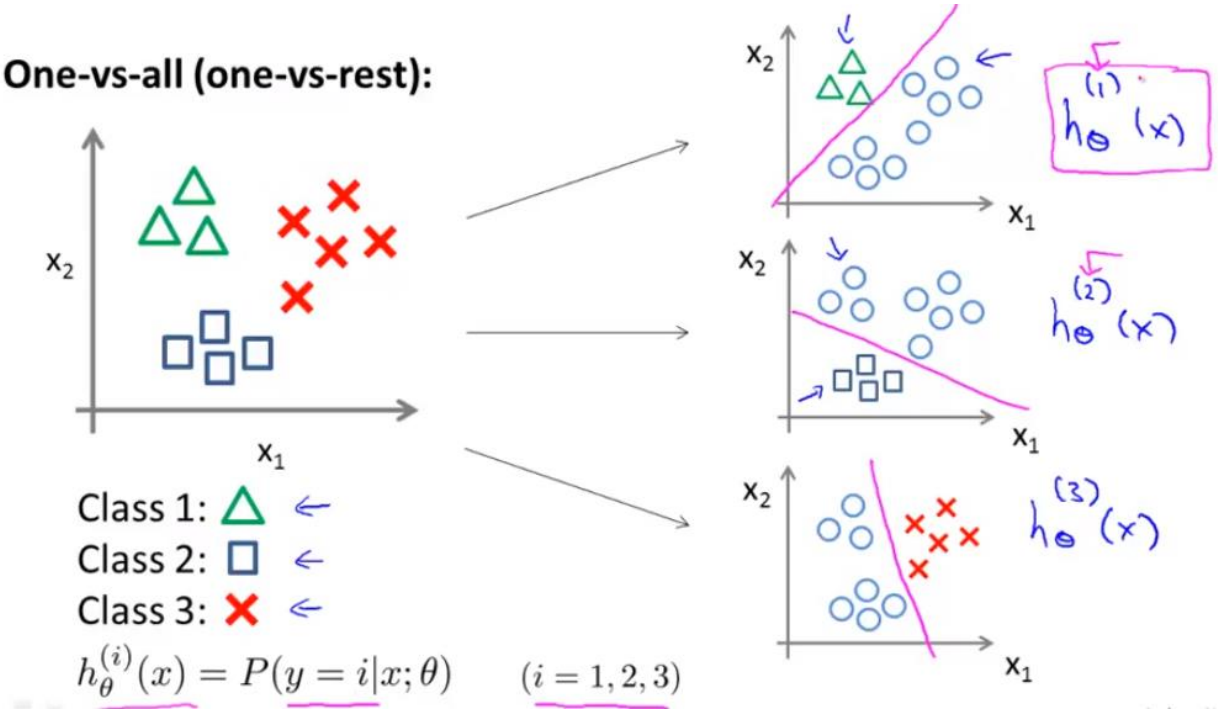
y=1 2 3

Weather: Sunny, Cloudy, Rain, Snow

y=1 2 3 4



- one versus all classification (one-vs-rest)
  - hypothesis를 class 개수 만큼 만들면 된다.



→ 각각의 hypothesis는 세모를, 네모를, x를 y=1로 취급하는 것이다.

- 어떤 x가 어떤 class에 속하는지 보기 위해서는 h(x)가 가장 크게 (1에 가깝게 만드는) hypothesis를 선택하면 된다. 그 hypothesis를 사용한 class를 선택하면 되는 것이다.

One-vs-all

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

$$y \in \{0, 1 \ldots n\}$$
$$h_\theta^{(0)}(x) = P(y = 0|x; \theta)$$
$$h_\theta^{(1)}(x) = P(y = 1|x; \theta)$$
$$\cdots$$
$$h_\theta^{(n)}(x) = P(y = n|x; \theta)$$
$$\text{prediction} = \max_i(h_\theta^{(i)}(x))$$

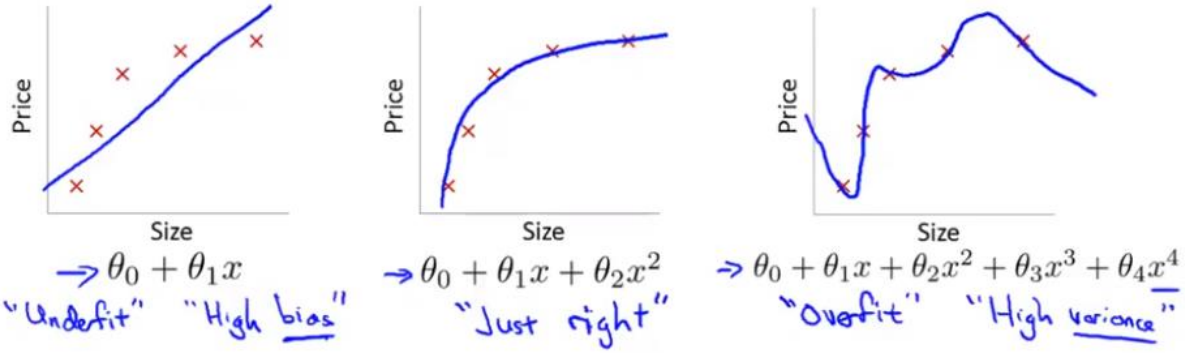
7. Application-3 : The problem of overfitting

7.1. 문제 :

예시1) linear regression

- linear regression의 경우에 → under-fitting = high bias
- quadratic → 적당함
- high order polynomial → overfitting = high variance

Example: Linear regression (housing prices)



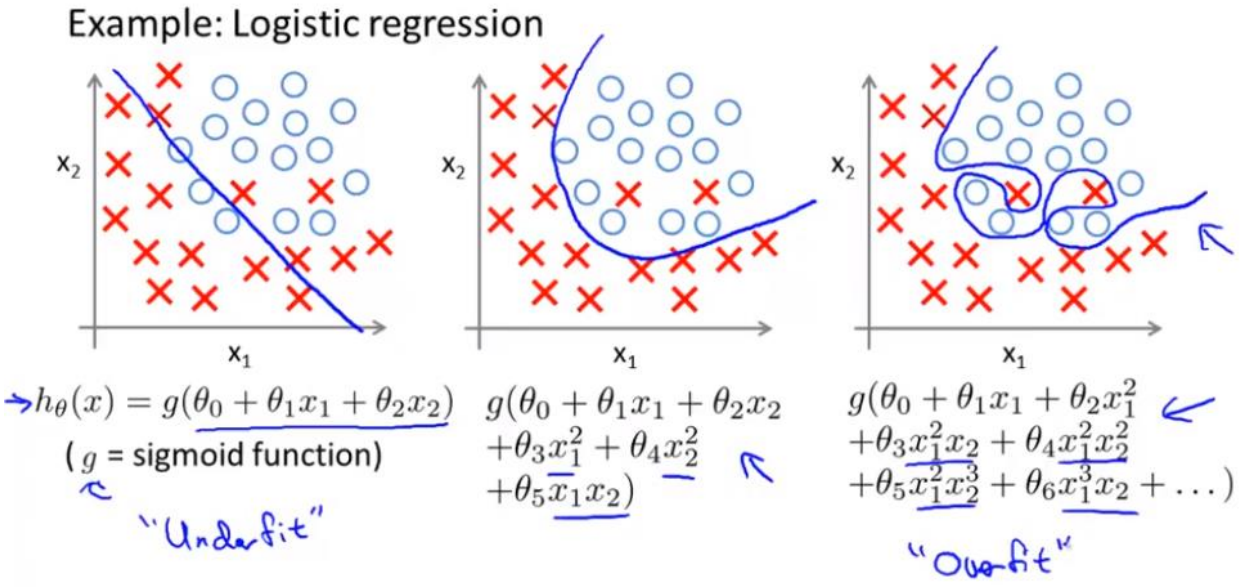
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

→ 이처럼 hypothesis가 data를 지나치게 fit할 때, 새로운 값들을 예측하는 게 어려울 수 있다. 그것이 overfitting

문제이다.

- 이 문제처럼 물론 degree of polynomial이 커질 때도(3번째 경우) 문제이지만, feature가 많아지는 경우에도 만약에 data set의 수가 적으면 문제가 발생할 수 있다. 구불구불한 것이 많아진다.

예시2) logistic regression



7.2. 해결 : regularization

첫번째는 feature를 다 쓰지 않고 빼는 방법이 있고, 두번째는 feature는 다 쓰면서 θ를 줄이는 방법(regularization)이 있다.

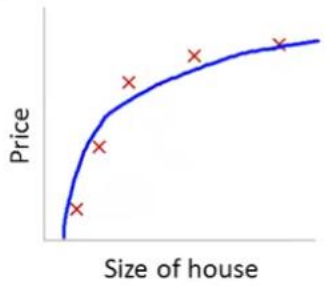
1. Reduce number of features.
   - Manually select which features to keep.
   - Model selection algorithm (later in course).
2. Regularization.
   - Keep all the features, but reduce magnitude/values of parameters θ_j.
   - Works well when we have a lot of features, each of which contributes a bit to predicting y.

(1) Reduce the number of features (with algorithm)

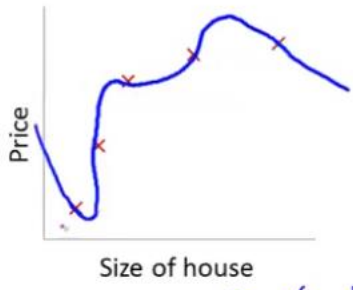
(2) Regularization

- 예시 : θ3와 θ4를 줄임으로써 x^3, x^4항을 없애는 효과를 낼 수 있다.

Intuition



$\theta_0 + \theta_1 x + \theta_2 x^2$



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

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

$\theta_3 \approx 0 \quad \theta_4 \approx 0$

→ small values for parameters (θ) leads to "simpler" "Smoother" hypothesis, less prone to overfitting

질문!!! 왜 저거를 굳이 더해버리는 거지? → minimize하려고

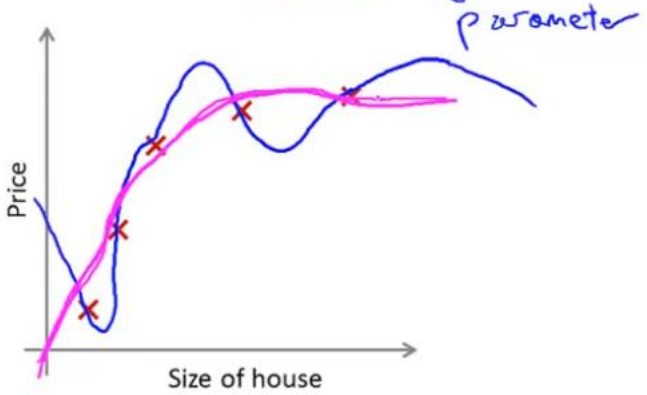
- 원리 :

Regularization.

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

regularization parameter

$$\min_\theta J(\theta)$$



→ θ를 모두 더한다. λ는 regularization parameter이고 θ의 합에 곱해서 regularization objective에 곱하는 것이다. 이 때, J가 두 항의 합으로 나타내지게 되는데, 앞의 항은 hypothesis랑 y의 차이를 줄여주는 효과를 내고, 뒤의 항은 parameter를 줄이는 효과를 내므로, 그것의 균형을 찾을 수 있게 된다.

왜 θ를 줄이면 이와 같이 굴곡이 사라지는가에 대한 직관은 직접 해봐야지 얻을 수 있다.

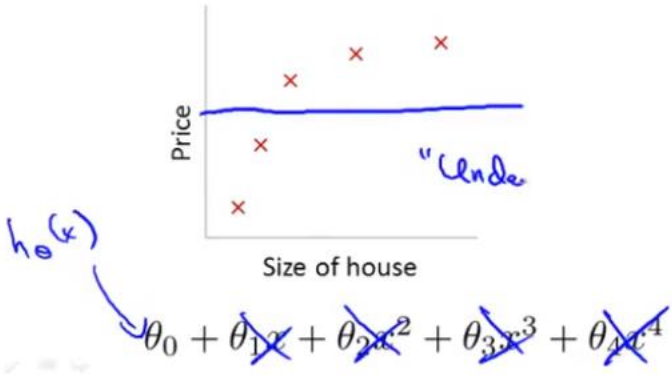
- λ (regularization parameter)의 조절 : λ를 높인다는 것은 θ가 높아지는 것에 대한 엄청난 패널티를 가한다는 것이다. 따라서 λ를 높이게 되면 θ가 0으로 가게 되고, 그렇게 되면 지나치게 parameter가 축소되어 모델이 단순화되는 문제점이 발생한다.

만약에 λ = 0이 되게 되면 반대로, 다시 overfitting의 문제가 살아날 것이다.

In regularized linear regression, we choose θ to minimize

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



$h_\theta(x)$

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

$\theta_1, \theta_2, \theta_3, \theta_4$

$\theta_1 \approx 0$, $\theta_2 \approx 0$

$\theta_3 \approx 0$, $\theta_4 \approx 0$

$h_\theta(x) = \theta_0$

- 미래 : non-linear classifiers

1.6. Neural Networks

1.6.1. Neural network를 언제 쓰는가.

- Neural network가 필요한 이유 : logistic regression 의 non-linear classification으로는 부족하기 때문이다

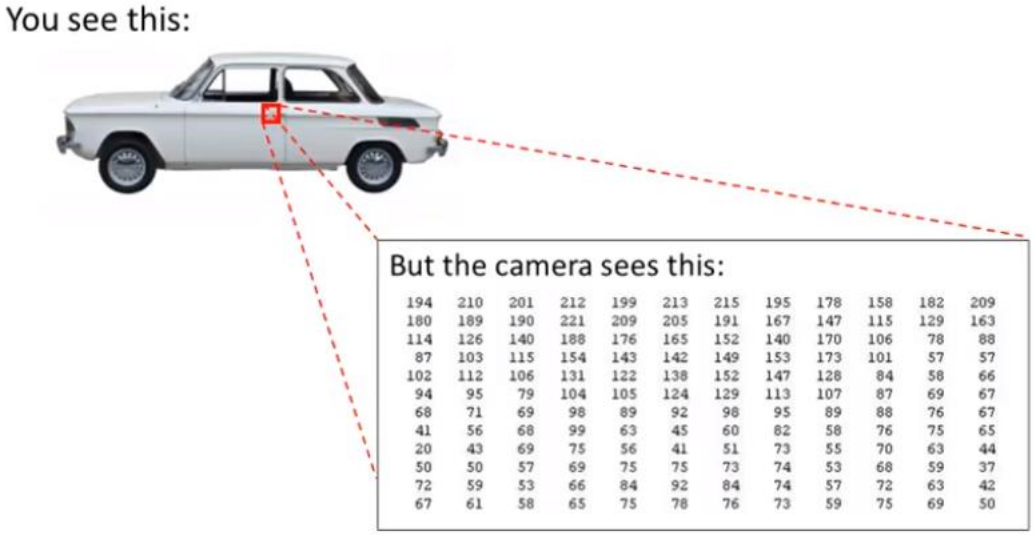
  (1) feature가 너무 많아지고, overfitting이 일어날 수 있고

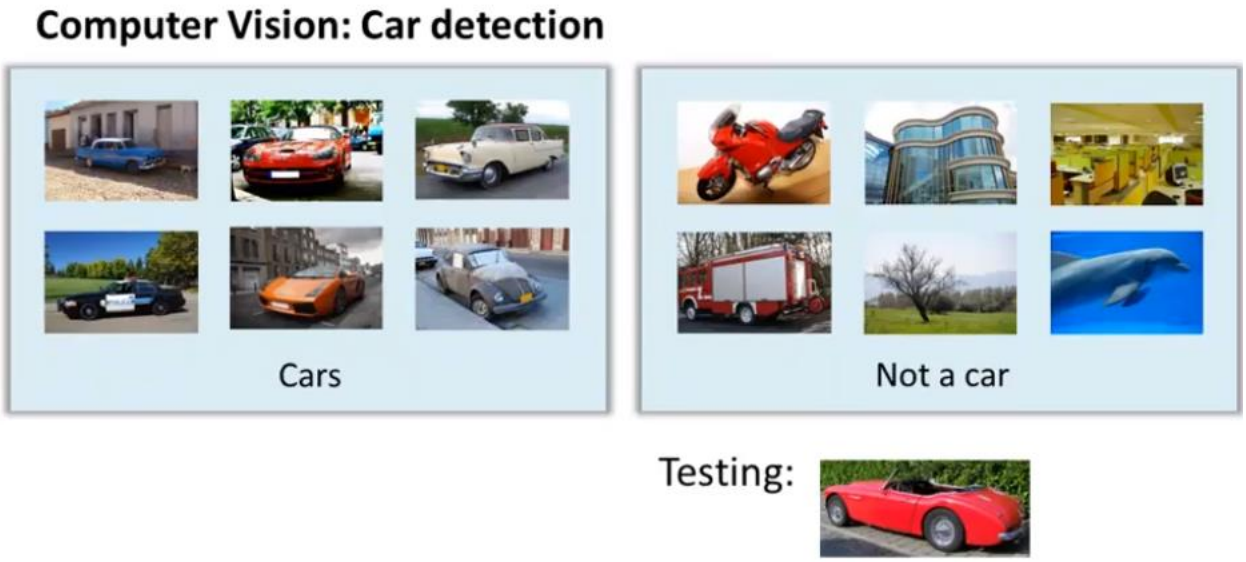
  (2) computationally expensive하다.

  → 더 fit가 잘 되는 것을 찾다보면 x1^2, x2^2, x3^2, 혹은 x1^3, x1^2*x2 등을 하다 보면 feature가 엄청 많아진다.

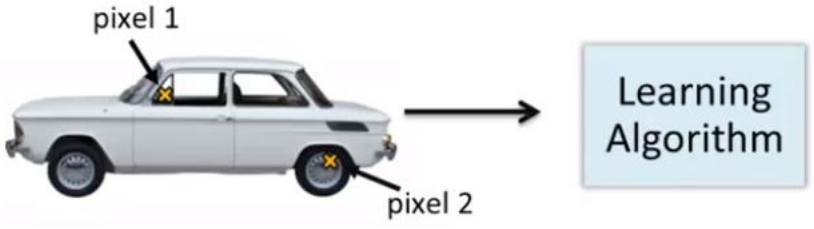
➔ why do we need neural network? : non-linear model을 더 잘 represent할 수 있기 때문이다.

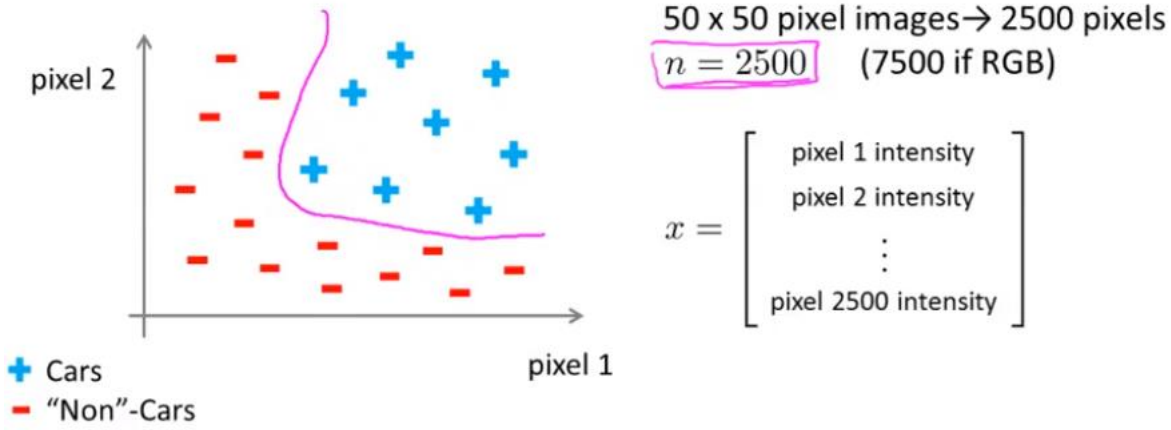
- 예시 : Computer vision





- non-linear classification으로 해보려는 경우!



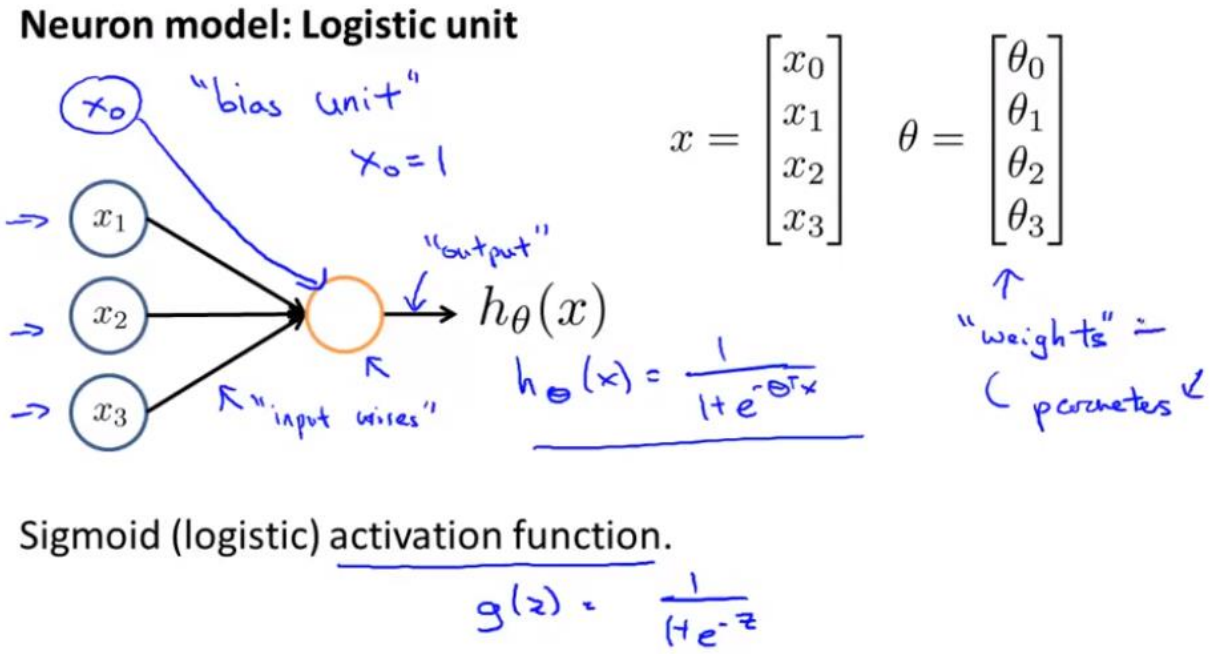


→ n이 크게 되면 변수가 너무 많아진다. 변수 2개를 곱한 형태는 n(n-1) / 2개이다.

- Neural network
  - mimicking the brain : 뉴런을 연결하면 뇌가 data를 배워서 어떻게 해석해야 하는지 알게 된다.

1.6.2. Neural network의 구조와 알고리즘

- Model representation :
  - input(dendrite) → output through Exon (activation의 과정)

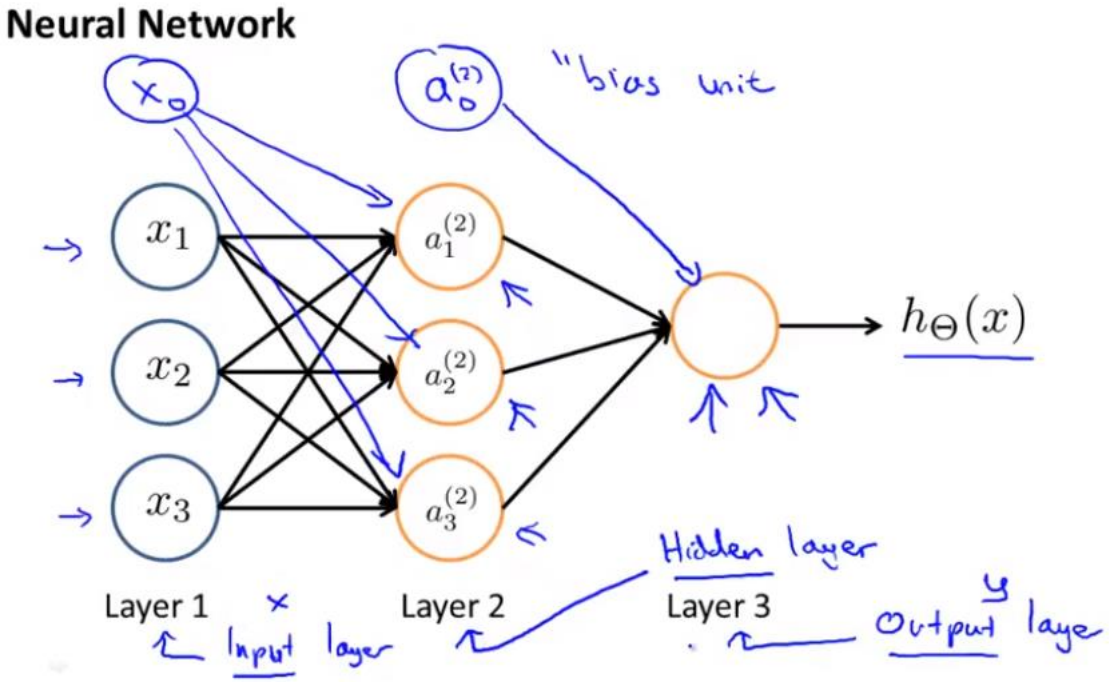


→ parameter가 weight라고 불린다.

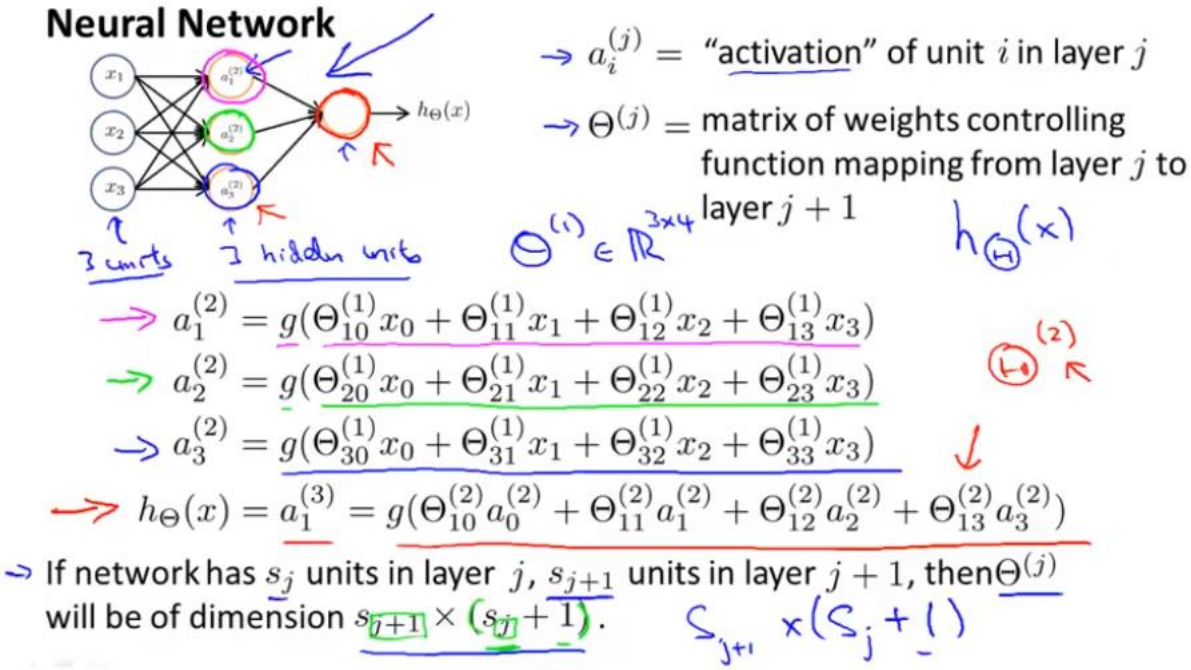
$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ matrix of weights controlling function mapping from layer j to layer $j+1$

- layer의 개수가 더 많아질 수 있다. : input과 output이 아닌 layer는 hidden layer라고 부른다. 1개 이상일 수도 있다. In this model our x0 input node is sometimes called the "bias unit." It is always equal to 1.



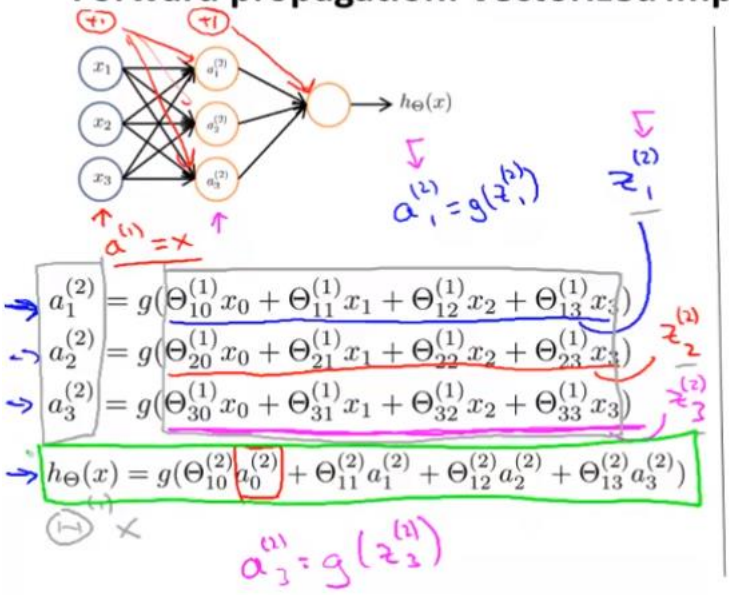
- Neural network의 알고리즘



→ θ 행렬이 x를 a로 만들어준다. θ 행렬의 dimension을 보게 되면, 일단 행의 개수는 다음 단계(j+1단계)의 변수의 개수여야 한다. 열의 개수는 그 전단계(j단계) + 1이어야 한다. 왜냐하면 bias unit → 1이 있기 때문이다.

- Vectorized implementation : vector로 만들어보자!!

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)}a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1.$

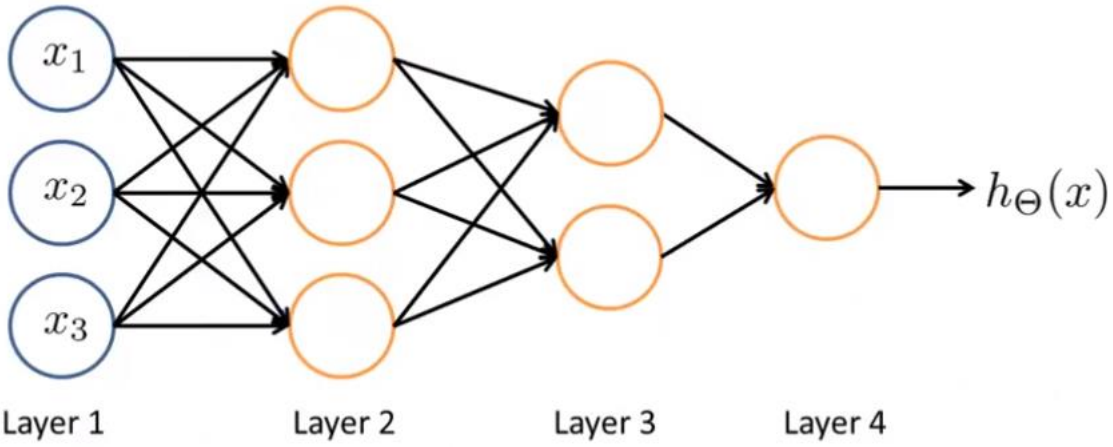
$$z^{(3)} = \Theta^{(2)}a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

$$z_k^{(2)} = \Theta_{k,0}^{(1)}x_0 + \Theta_{k,1}^{(1)}x_1 + \cdots + \Theta_{k,n}^{(1)}x_n$$

$$a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$

→

Other network architectures



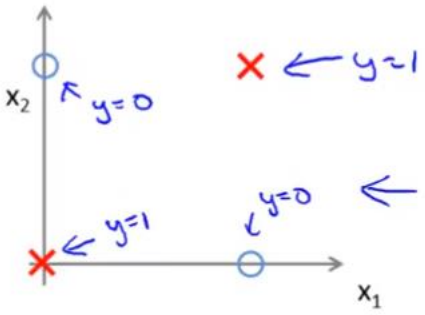
- x1, x2, x3가 θ행렬로 가공되어 a1, a2, a3가 나온다. → between layer j and layer j+1, we are doing exactly the same thing as we did in logistic regression.

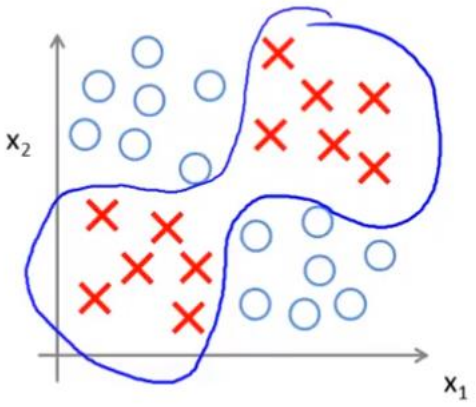
$$z^{(2)} = \Theta^{(1)}a^{(1)};\ a^{(2)} = g(z^{(2)})$$

- Example :

Non-linear classification example: XOR/XNOR

→ x_1, x_2 are binary (0 or 1).





$y = x_1 \text{ XOR } x_2$

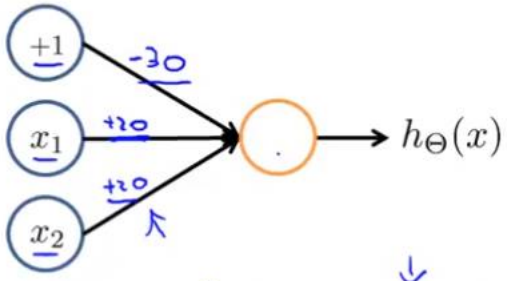
$x_1 \text{ XNOR } x_2$

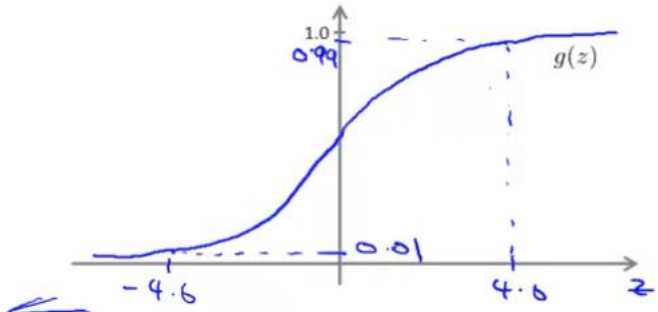
$\text{NOT } (x_1 \text{ XOR } x_2)$

Simple example: AND

→ $x_1, x_2 \in \{0, 1\}$

→ $y = x_1 \text{ AND } x_2$





→ $h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

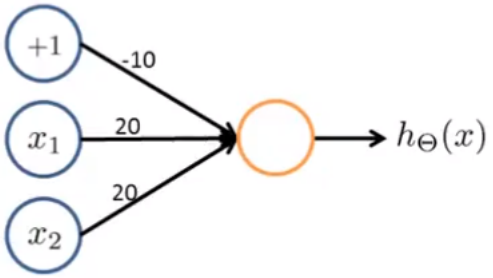
$\Theta_{10}^{(1)}$ $\Theta_{11}^{(1)}$ $\Theta_{12}^{(1)}$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
→ 0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
→ 1	1	$g(10) \approx 1$

$h_\Theta(x) \approx x_1 \text{ AND } x_2$

- Or function

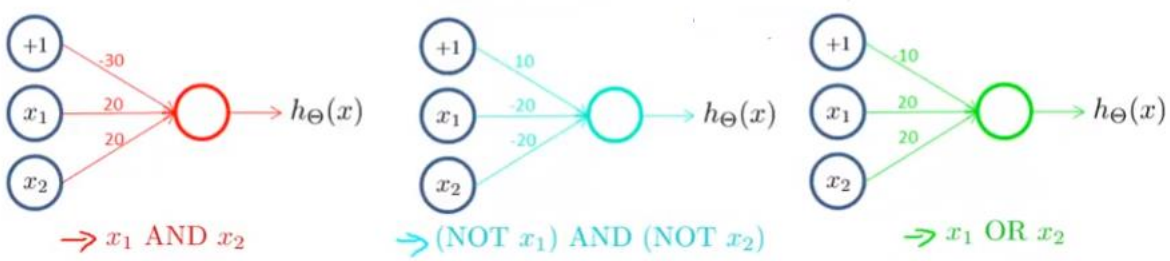
Example: OR function



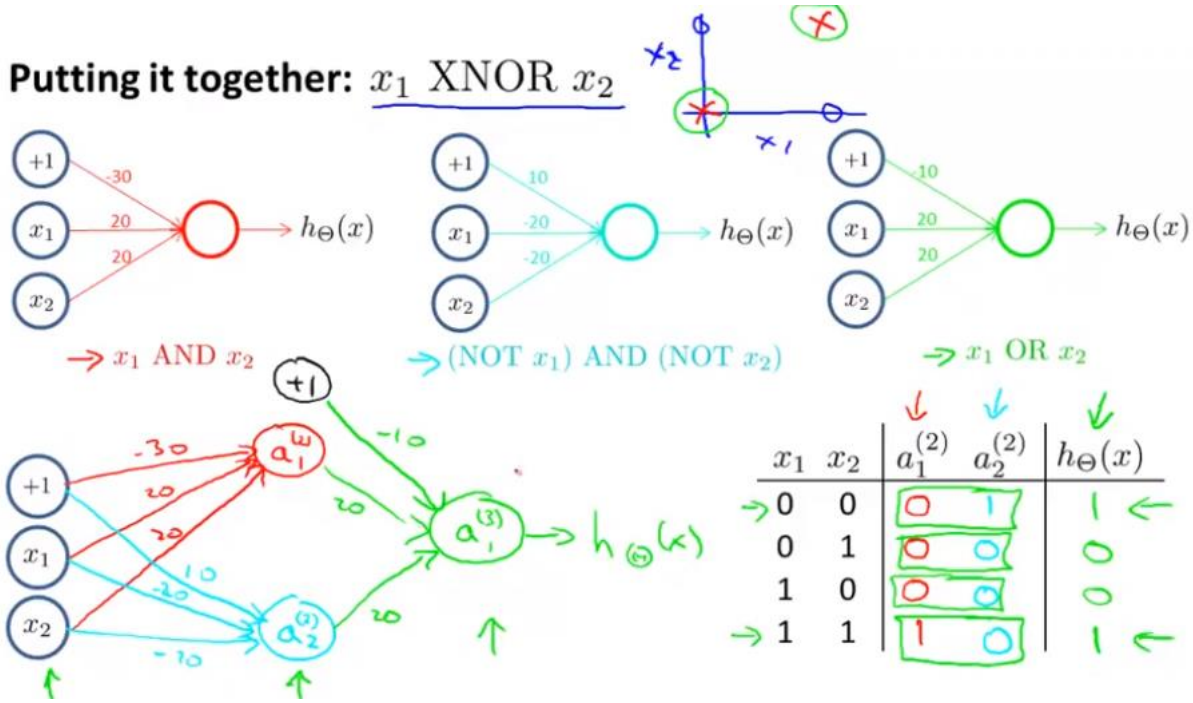
$g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	≈ 1
1	1	≈ 1

- Other example



- XNOR function을 만들어보자! 둘다 0이거나, 둘다 1이거나. (exclusive nor이다)
- XOR은 exclusive or → (1,0), (0,1) 만 true



→ 여기에는 hidden layer도 있다.

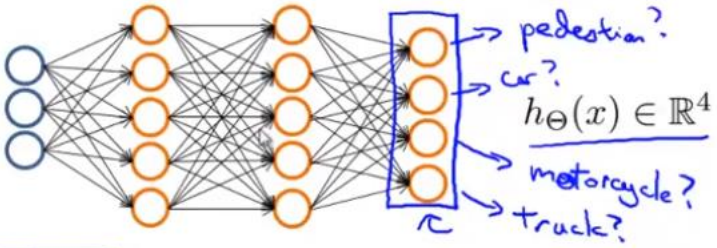
→ 이게 어떻게 가능한거지?? → multiclass classification

→ 예를 들어서 숫자를 삼등분해서 ...

- Multiclass classification
  - 아까는 0아니면 1이 h(x)로의 전부였으나, 이번에는 결과 output이 정말 많다.

Multiple output units: One-vs-all.

Pedestrian Car Motorcycle Truck



Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

Previously $y \in \{1, 2, 3, 4\}$

$h_\Theta(x^{(i)}) \approx y^{(i)}$

- 구조 : → hidden layer를 저절로 만들어준다.

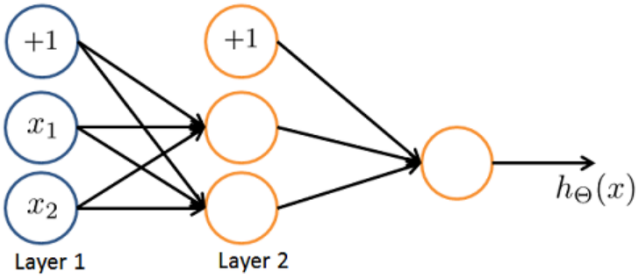
$$\begin{bmatrix}x_0\\x_1\\x_2\\\cdots\\x_n\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(2)}\\a_1^{(2)}\\a_2^{(2)}\\\cdots\end{bmatrix} \rightarrow \begin{bmatrix}a_0^{(3)}\\a_1^{(3)}\\a_2^{(3)}\\\cdots\end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix}h_\Theta(x)_1\\h_\Theta(x)_2\\h_\Theta(x)_3\\h_\Theta(x)_4\end{bmatrix}$$

- 질문 :

✔

1 / 1 points

5. You are using the neural network pictured below and have learned the parameters $\Theta^{(1)} = \begin{bmatrix}1 & -1.5 & 3.7\\1 & 5.1 & 2.3\end{bmatrix}$ (used to compute $a^{(2)}$) and $\Theta^{(2)} = [1 \quad 0.6 \quad -0.8]$ (used to compute $a^{(3)}$) as a function of $a^{(2)}$). Suppose you swap the parameters for the first hidden layer between its two units so $\Theta^{(1)} = \begin{bmatrix}1 & 5.1 & 2.3\\1 & -1.5 & 3.7\end{bmatrix}$ and also swap the output layer so $\Theta^{(2)} = [1 \quad -0.8 \quad 0.6]$. How will this change the value of the output $h_\Theta(x)$?



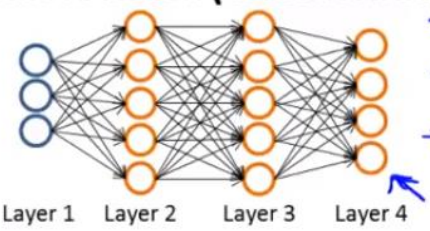
→ 왜 이게 답이 바뀌지 않는다일까? 순서가 반대가 되니까 당연히 같지!!

1.6.3. Neural Network의 설계 방법 : How to minimize cost function

→ Backpropagation /

- Multiclass classification

Neural Network (Classification)



→ $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

→ $L =$ total no. of layers in network $L = 4$

→ $s_l =$ no. of units (not counting bias unit) in layer l $s_1 = 3,\ s_2 = 5,\ s_4 = s_L = 4$

Binary classification

$y = 0$ or 1 ←

$h_\Theta(x)$

1 output unit ←

$h_\Theta(x) \in \mathbb{R}$

$s_L = 1$, $K = 1$

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$ ←

pedestrian car motorcycle truck

K output units

$h_\Theta(x) \in \mathbb{R}^K$

$s_L = K$ $(K \geq 3)$

Let's first define a few variables that we will need to use:

- L = total number of layers in the network
- s_l = number of units (not counting bias unit) in layer l
- K = number of output units/classes

→ logistic regression에서 sample x의 training set 데이터가 m개 있으니까 1부터 m까지 시그마로 하는거다. 다만 이제 Neural Network로 하게 되면 y값이 0,1의 하나의 값이 아니라 벡터로 존재하다 보니까 cost function도 i = 1부터 k까지 다 더해야 한다.

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Neural network:

→ $h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th}$ output

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

→ regularization term에서는 x0 부분(bias term)을 하지 않는다.

- regularization term 관련

Note:

- the double sum simply adds up the logistic regression costs calculated for each cell in the output layer
- the triple sum simply adds up the squares of all the individual Θs in the entire network.
- the i in the triple sum does **not** refer to training example i

- How to minimize cost function of Neural network

1.6.3.1. Backpropagation

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

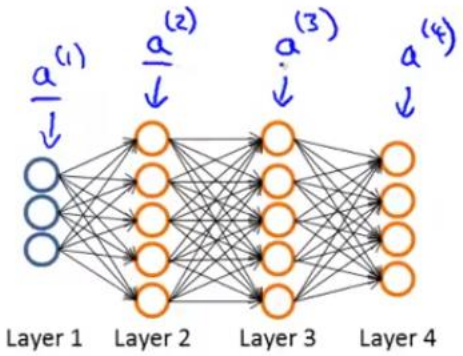
→ l 은 layer다.

Gradient computation

Given one training example (x, y):

Forward propagation:

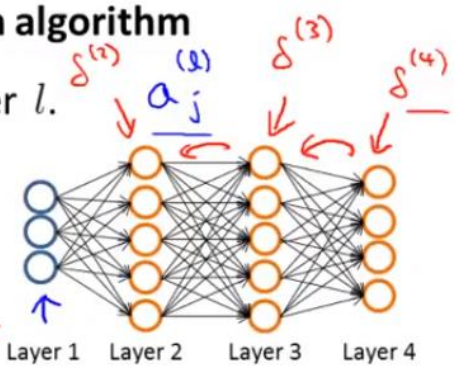
$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



- cost function을 편미분하려면 backpropagation을 하는 방법이 있다. 델타4를 구하고, 대입하여 델타3를 구하고, 델타2를 구하게 되면 거기다가 각각 activation을 곱하면 cost function을 편미분한 값이 된다.
이거 미분이 왜 저렇게 되는거지?

Gradient computation: Backpropagation algorithm

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l.



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

(No $\delta^{(1)}$)

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$ (ignore λ; if $\lambda = 0$)

빨간 색은 vectorized implementation

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j). (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m ← $(x^{(i)}, y^{(i)})$

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ ← $\Delta^{(2)} := \Delta^{(2)} + \delta^{(2+1)} (a^{(2)})^T$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

- 정리

Back propagation Algorithm

Given training set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$

- Set $\Delta_{i,j}^{(l)}$:= 0 for all (l,i,j), (hence you end up having a matrix full of zeros)

For training example t =1 to m:

1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for l=2,3,…,L

3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$

Where L is our total number of layers and $a^{(L)}$ is the vector of outputs of the activation units for the last layer. So our "error values" for the last layer are simply the differences of our actual results in the last layer and the correct outputs in y. To get the delta values of the layers before the last layer, we can use an equation that steps us back from right to left:

4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \, .* \, a^{(l)} \, .* \, (1 - a^{(l)})$

The delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l. We then element-wise multiply that with a function called g', or g-prime, which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$.

The g-prime derivative terms can also be written out as:

$$g'(z^{(l)}) = a^{(l)} \, .* \, (1 - a^{(l)})$$

5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

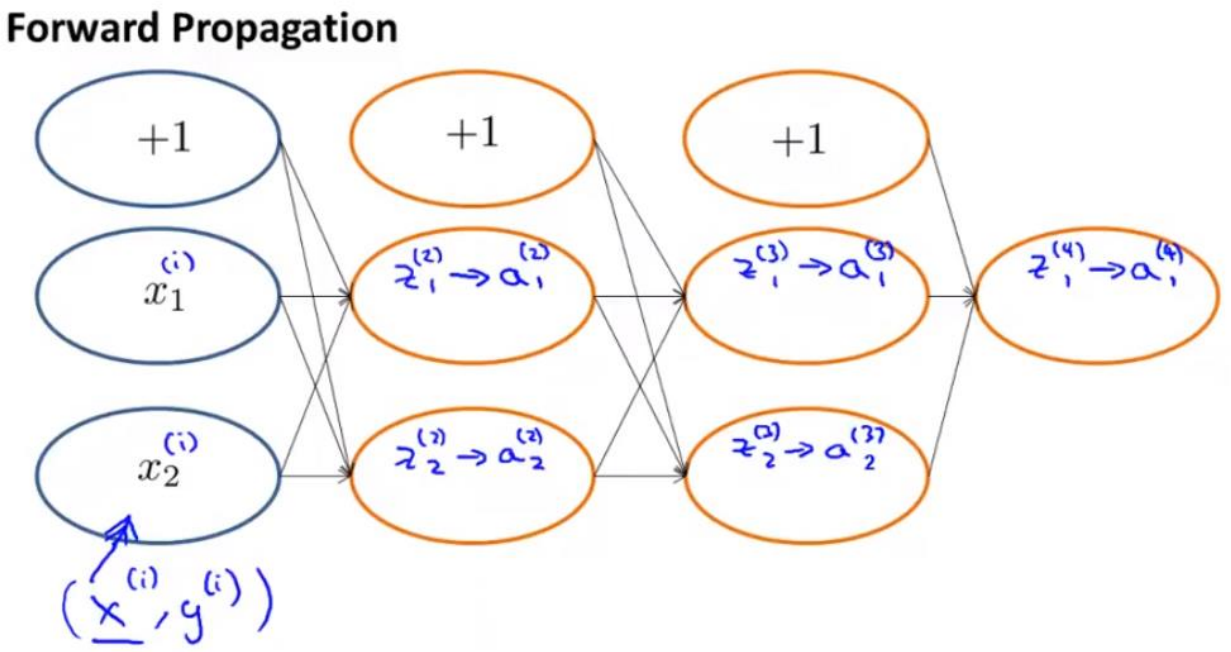
Hence we update our new Δ matrix.

- $D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right)$, if j≠0.
- $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$ If j=0

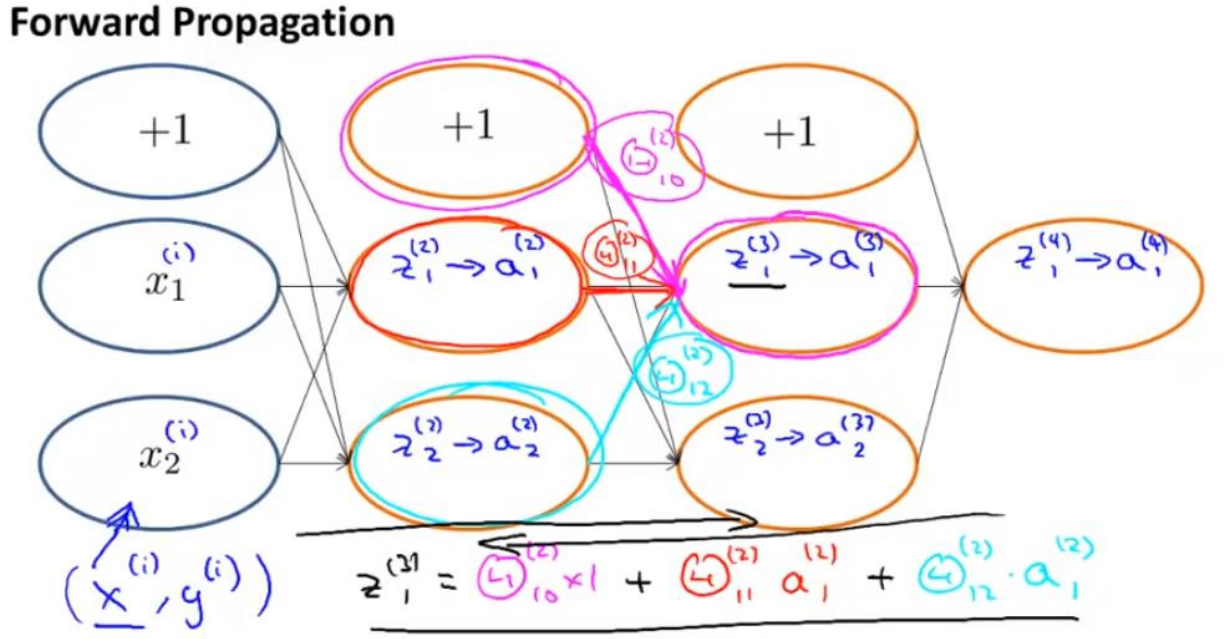
The capital-delta matrix D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

- 이 과정이 이해가 안되더라도 괜찮다. 어려운 건 누구에게나 마찬가지다. 블랙박스 같지만, 순차적으로 차근차근 이해하도록 해보자.

- Forward Propagation의 과정 :



→ 이렇게 흘러간다. 좀 더 자세히 $Z1^{(3)}$이 어떻게 만들어지는지 살펴보자.



- Backpropagation

What is backpropagation doing?

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1-y^{(i)}) \log(1-(h_\Theta(x^{(i)})))\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

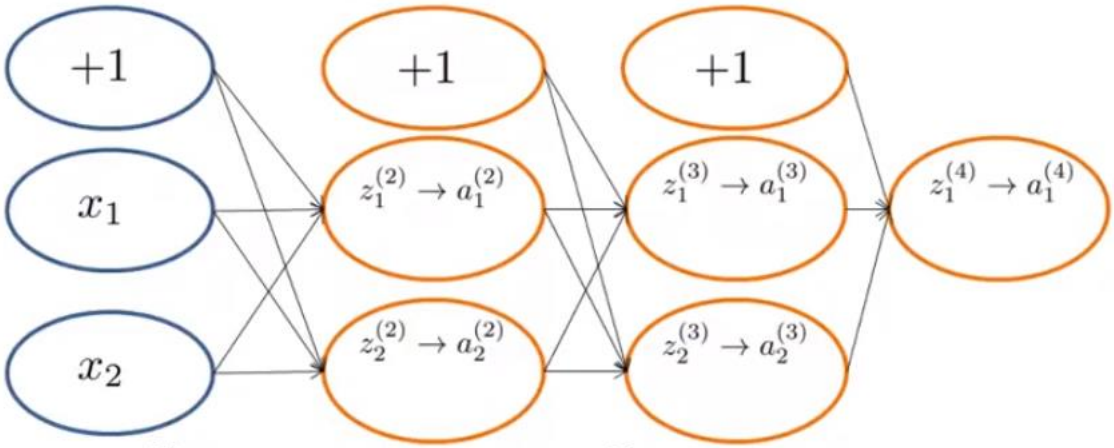
Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log h_\Theta(x^{(i)})$$

(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

I.e. how well is the network doing on example i?

Forward Propagation

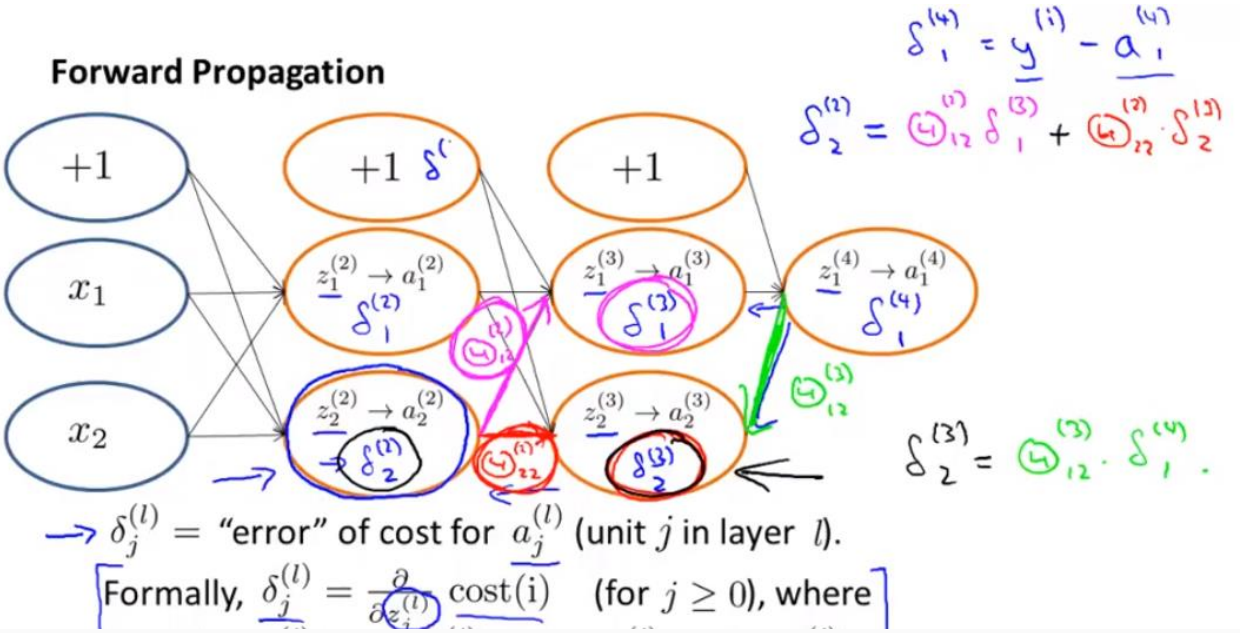


→ $\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost(i)}$ (for $j \geq 0$), where

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log h_\Theta(x^{(i)})$$

→ 이 편미분식은 이해가 안 된다. 어쨌든 간에 δ(4)를 구하고 δ(3)를 구하고, 거꾸로 구하는 방법이다.



If we consider simple non-multiclass classification (k = 1) and disregard regularization, the cost is computed with:

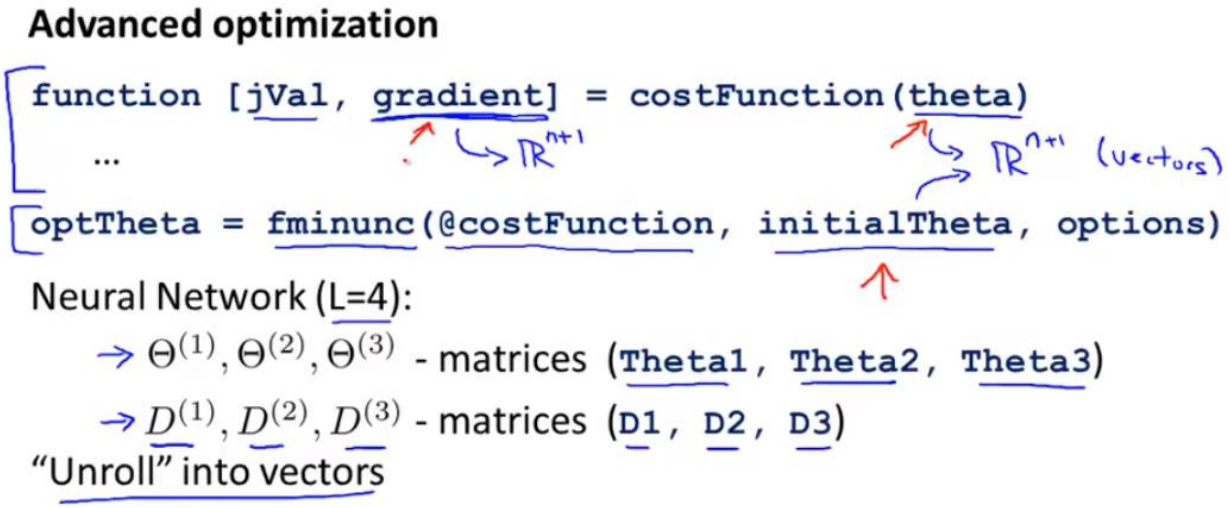
$$cost(t) = y^{(t)} \log(h_\Theta(x^{(t)})) + (1 - y^{(t)}) \log(1 - h_\Theta(x^{(t)}))$$

→ δ(2) = theta * δ(3) + theta * δ(3)라는 게 어떻게 나온거지? 왜 이렇게 계산을 해야만 하는거지? 굉장히 신기하다.

1.6.4. Backpropagation과 octave implementation - Octave에서 어떻게 해야하는가!!

1.6.4.1. Unrolling parameters

- 목적 : optimization
- 이제는 theta가 vector가 아니다. matrices이다.



→ 이 matrix data를 어떻게 unroll해야지 빨간색으로 있는 gradient나 theta에 vector 형태로 집어넣을 수 있을까?

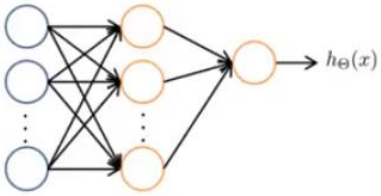
- (:) 를 쓰고 다시 행렬로 바꿀 때는 **reshape** command를 쓰면 된다.

Example

$s_1 = 10, s_2 = 10, s_3 = 1$

→ $\Theta^{(1)} \in \mathbb{R}^{10\times 11}, \Theta^{(2)} \in \mathbb{R}^{10\times 11}, \Theta^{(3)} \in \mathbb{R}^{1\times 11}$

→ $D^{(1)} \in \mathbb{R}^{10\times 11}, D^{(2)} \in \mathbb{R}^{10\times 11}, D^{(3)} \in \mathbb{R}^{1\times 11}$




```
→ thetaVec = [ Theta1(:); Theta2(:); Theta3(:)];
→ DVec = [D1(:); D2(:); D3(:)];

  Theta1 = reshape(thetaVec(1:110),10,11);
→ Theta2 = reshape(thetaVec(111:220),10,11);
↪ Theta3 = reshape(thetaVec(221:231),1,11);
```

→ ; 로 연결하니까 열로 쭉쭉 길게 unroll할 수 있다.

- 이것을 활용해보자. 이것은 gradient descent가 아니라 advanced optimization method이다.

Learning Algorithm

→ Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.

→ Unroll to get `initialTheta` to pass to

→ `fminunc(@costFunction, initialTheta, options)`

```
function [jval, gradientVec] = costFunction(thetaVec)
```

→ From `thetaVec`, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$. reshape

→ Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$.

Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get `gradientVec`.

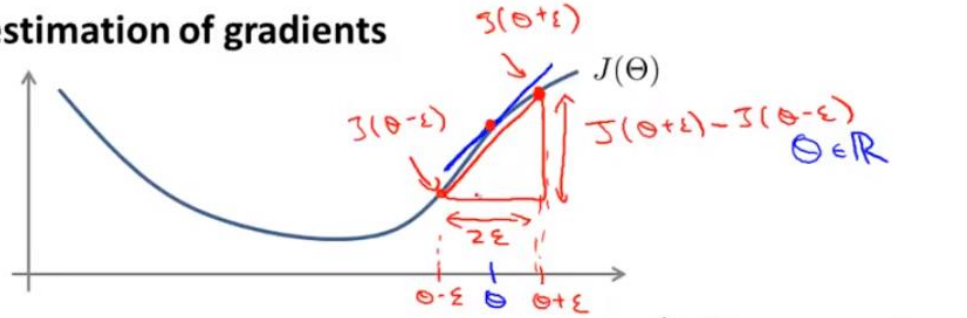
→ initial theta를 만들 때는 vector로 길게 해서 집어넣고, reshape한 다음에 D 계산하고(reshape해서 행렬로 만들어야 Forward, Backpropagation이 편하다), D를 다시 vector로 해서 gradientVec에 assign한다.

- 문제 : backpropagation이 잘 되는 것 같아 보여도 잘 못 된 것이 있을 수 있다!

➔ Gradient checking

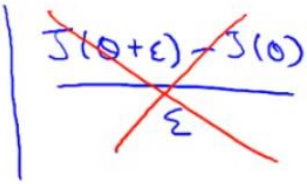
- Cost function J 한점의 gradient를 구하기 위해서 함수 위의 두 점을 이어서 기울기를 구한다.
- 계산 방법 :

Numerical estimation of gradients



$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

$\varepsilon = 10^{-4}$



Implement:
```
gradApprox = (J(theta + EPSILON) - J(theta - EPSILON))
             /(2*EPSILON)
```

→ 이를 다시 쓰면 다음과 같다. epsilon은 10 ^-4가 적당하다.

Parameter vector θ

→ $\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)

→ $\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$

→ $\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \ldots, \theta_n)}{2\epsilon}$

→ $\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \ldots, \theta_n)}{2\epsilon}$

$\vdots$

→ $\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n+\epsilon) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n-\epsilon)}{2\epsilon}$

- 이를 backpropagation의 D와 얼마나 같은지 비교하면 된다.

```
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) = thetaPlus(i) + EPSILON;
  thetaMinus = theta;
  thetaMinus(i) = thetaMinus(i) - EPSILON;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                  /(2*EPSILON);
end;
```

$\frac{\partial}{\partial \theta_i} J(\theta)$.

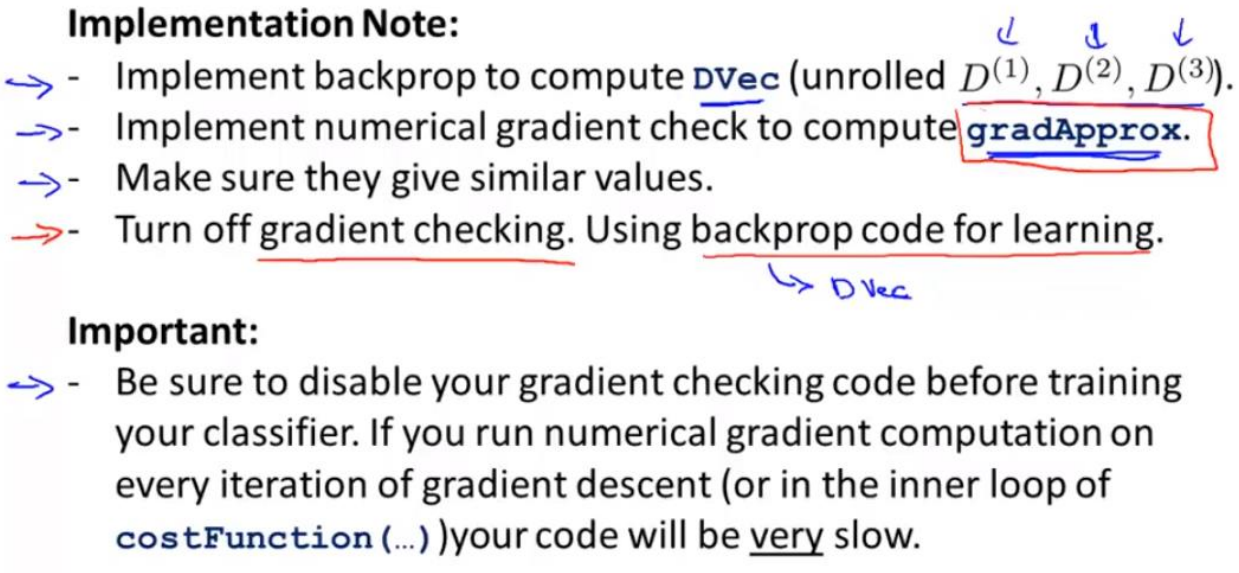
$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \varepsilon \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \theta_i - \varepsilon$

Check that `gradApprox` ≈ `DVec`

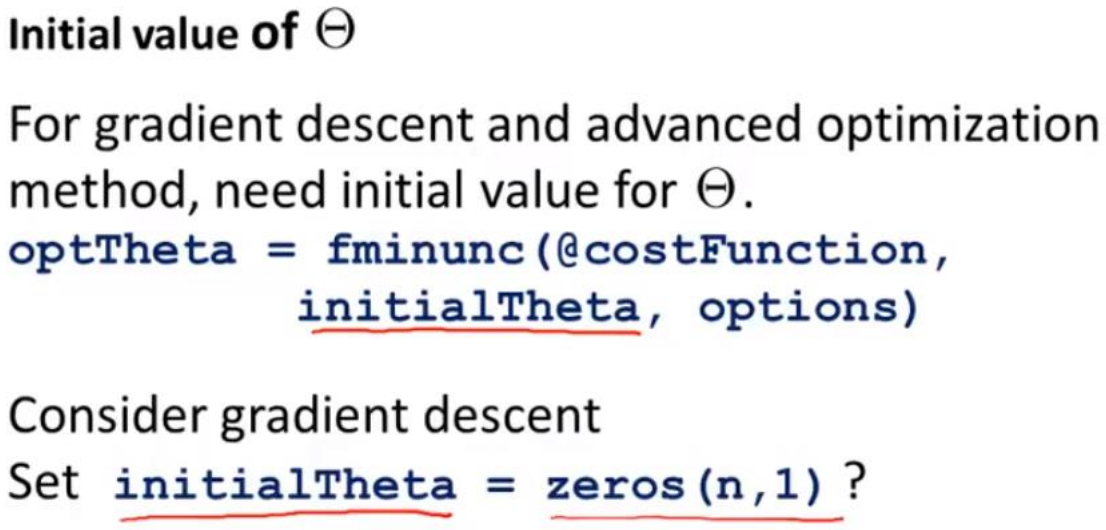
From backprop.

```
epsilon = 1e-4;
for i = 1:n,
  thetaPlus = theta;
  thetaPlus(i) += epsilon;
  thetaMinus = theta;
  thetaMinus(i) -= epsilon;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)
end;

```

- 실행 방법 : gradApprox는 computationally expensive하기 때문에, 맞는지만 확인하고 꺼야 한다.



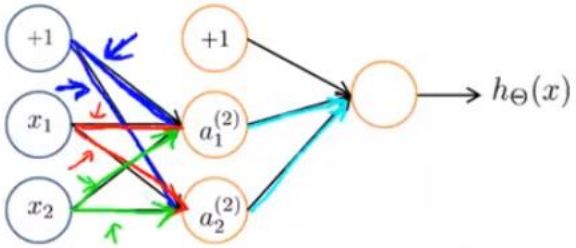
1.6.4.2. Random Initialization - 어떻게 initial Theta를 정할 것인가?



- theta를 모두 zero로 하면 안 되는 이유:

→ hidden unit a가 다 똑같이 나와서(a1=a2), theta를 업데이트하려고 해도 잘 안된다!! Backpropagation을 한다고 가정을 하면, a가 같게 되면 δ도 같게 된다. δ가 같게 되면 cost function도 같게 된다. 따라서 cost function을 미분한 값이 같게 되어서 거기에 learning rate를 곱한 값으로 gradient descent를 해도 theta값이 다르지 않게 된다.

Zero initialization



$\rightarrow \Theta_{ij}^{(l)} = 0$ for all i, j, l.

$a_1^{(2)} = a_2^{(2)}$. Also $\delta_1^{(2)} = \delta_2^{(2)}$.

$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$ $\quad \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

After each update, parameters corresponding to inputs going into each of two hidden units are identical.

→ 질문!! theta가 다 0이라면서? 왜 꼭 2개씩 짝을 이루어서 같아야 하는 것인가?

- 해결 방법 : symmetry breaking

Random initialization: Symmetry breaking

→ Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g. random 10×11 matrix (betw. 0 and 1)

```
→ Theta1 = rand(10,11)*(2*INIT_EPSILON)
           - INIT_EPSILON;

→ Theta2 = rand(1,11)*(2*INIT_EPSILON)
           - INIT_EPSILON;
```

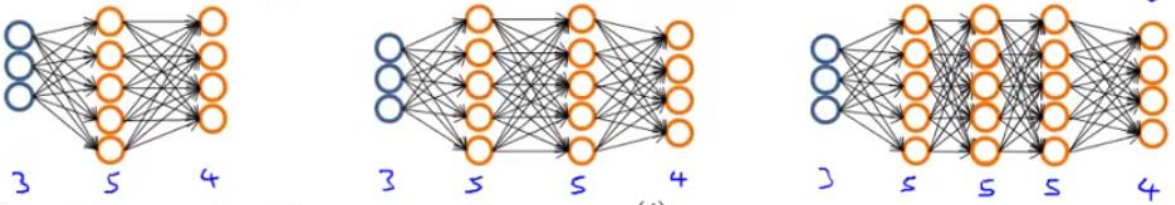
$[-\epsilon, \epsilon]$

1.6.5. Neural network 종합

Step1) Pick a neural network architecture!

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



→ No. of input units: Dimension of features $x^{(i)}$

→ No. output units: Number of classes

→ hidden layer를 1개 넣는게 가장 좋다. 2개 이상이면, 각 hidden layer에 unit 수가 layer끼리 같으면 좋다.

hidden layer의 unit 수가 많으면 좋다.

Step 2) Training a neural network

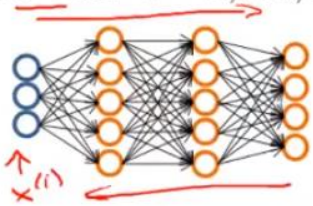
Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

for i = 1:m {

Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).



Step3) Gradient checking (검토) / Optimation mothod

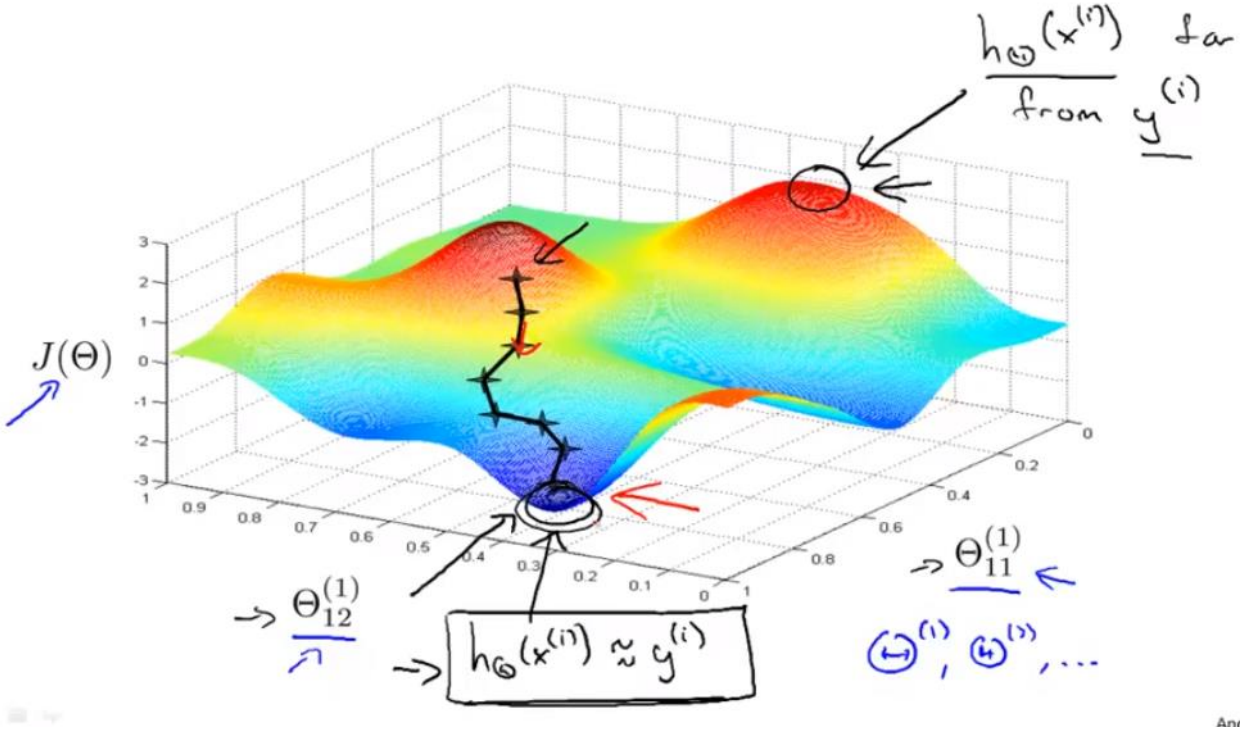
Training a neural network

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed us backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.

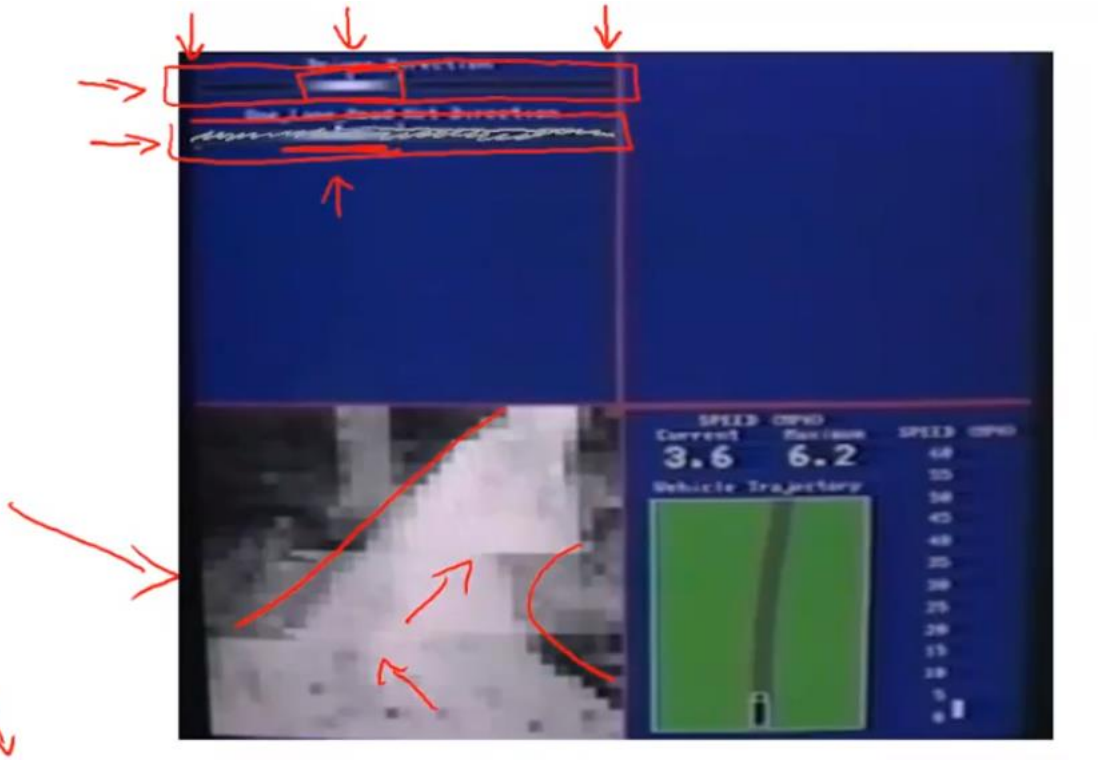
   Then disable gradient checking code.
6. Use gradient descent or advanced optimization method w backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

→ Gradient descent가 좋다.

ex) gradient descent의 직관



1.6.6. 자율 주행에의 응용



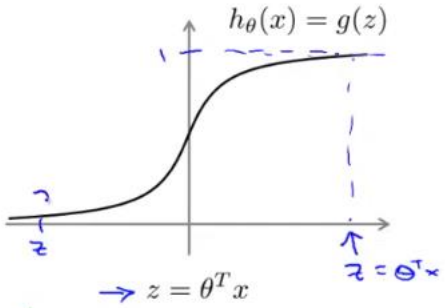
→ 왼쪽 위의 위치가 steering wheel의 위치이다. 위의 bar는 사람이 운전대를 어떻게 잡는지. 주행을 할 때, 사람이 하는 것을 기록한다. 이미지와 함께.

1.7. Supported vector machine - An alternative for logistic regression ← supervised learning

- SVM 개괄 : supervised learning에서 logistic regression의 응용 버전이라고 생각하면 편하다.
- Logistical regression의 복습

Alternative view of logistic regression

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



If $y = 1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$
If $y = 0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

→ 여기서 y=1, y=1일 때, cost는 이렇게 된다.

1.7.1. What is SVM?

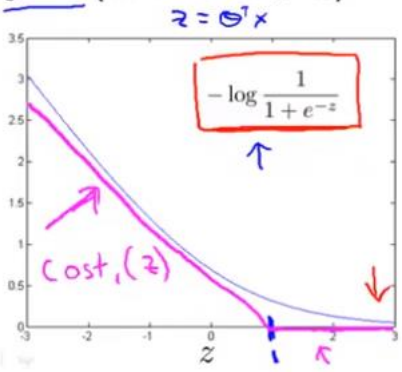
- 목적 : cost function을 간략히 직선화해서 computational cost 줄이는 효과

Alternative view of logistic regression

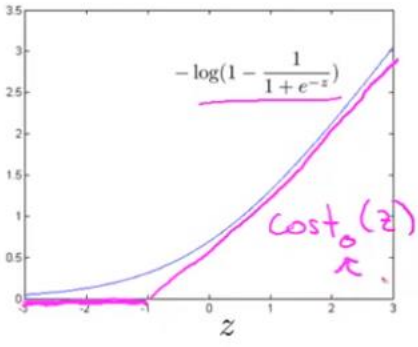
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

If $y = 1$ (want $\theta^T x \gg 0$):



If $y = 0$ (want $\theta^T x \ll 0$):



- logistic regression의 변형 : cost function을 직선 2개로 표현하게끔 바꾼 것이다. Cost function의 약간의 변형이 있다.

변형-(1) log 함수를 직선의 함수로 바꾼다.

변형-(2) min 함수에 어떤 수를 곱해도 그것을 최소화하는 θ는 달라지지 않기 때문에 → 관습적으로 regularization term의 lambda를 없애고 cost function 앞에 C 라는 상수를 도입한다. (따라서, C = 1/λ 이므로 C를 크게 하는 것은 λ를 작게 하는 것과 같은 효과이다.)

Support vector machine

Logistic regression:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$\text{cost}_1(\theta^T x^{(i)})$ $\text{cost}_0(\theta^T x^{(i)})$ A B

Support vector machine:

$$\min_\theta C\sum_{i=1}^{m} y^{(i)}\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

$\min_u ((u-5)^2+1) \rightarrow u=5$ ×10

$\min_u 10(u-5)^2+10 \rightarrow u=5$

$A + \lambda B$ ← $C = \frac{1}{\lambda}$

$CA + B$ ←

$$\therefore \min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

-

SVM hypothesis

$$\Rightarrow \min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Hypothesis:

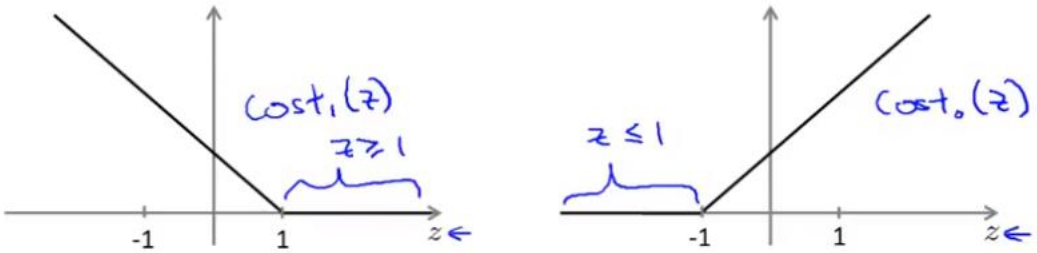
$$h_\theta(x) \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

1.7.2. Why do we use SVM? - 이것을 왜 쓰는가? Large margin classifier이기 때문에!!

- implements extra safety margin factor → θx가 확실하게 1을 넘기를 원한다.

Support Vector Machine

$$\rightarrow \min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



→ If $y=1$, we want $\theta^T x \geq 1$ (not just ≥ 0) $\theta^T x \geq 1$

→ If $y=0$, we want $\theta^T x \leq -1$ (not just < 0) $\theta^T x \leq -1$

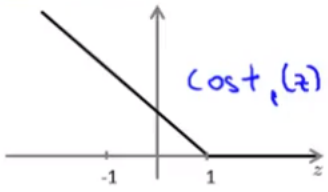
- C가 엄청 크면 cost function이 0에 가까워서 minimize될 것이다.

SVM Decision Boundary

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Whenever $y^{(i)} = 1$:

$\theta^T x^{(i)} \geq 1$



Whenever $y^{(i)} = 0$:

$\theta^T x^{(i)} \leq -1$

이 식을 최소화하게 되면 cost function의 앞부분은 완전히 0이 되어 버린다. 식을 다시 써보자.

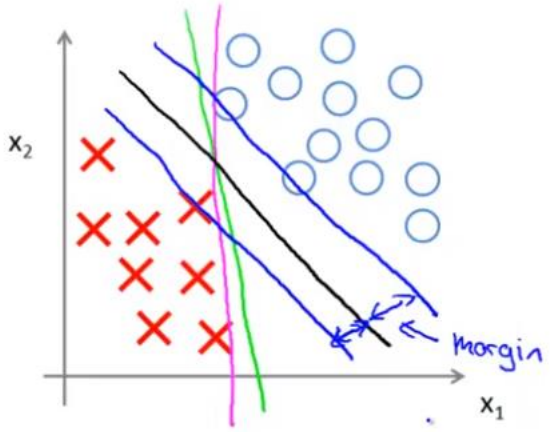
$$\min \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

$$s.t.\quad \theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$$

$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0.$$

- Consequence : 그렇게 되면 large margin을 갖게 된다. (이 부분이 명확한 설명이 없다.) margin이란 decision boundary 선으로부터의 data point의 거리

SVM Decision Boundary: Linearly separable case

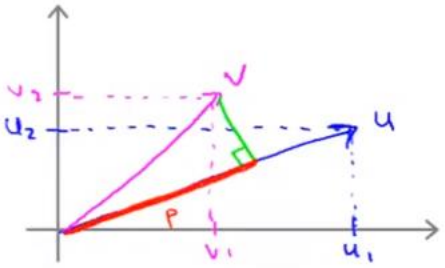


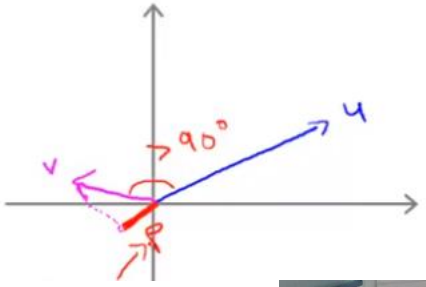
Large margin classifier

- margin을 극대화하는 수학적 증명 :

1) inner product가 정사된 길이 곱하기 V 벡터의 길이라는 것을 증명하기.

Vector Inner Product





$\rightarrow u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \rightarrow v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$u^T v = ?$ $\quad [u_1 \ u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$\|u\|$ = length of vector u

$= \sqrt{u_1^2 + u_2^2} \quad \in \mathbb{R}$

p = length of projection of v onto u.

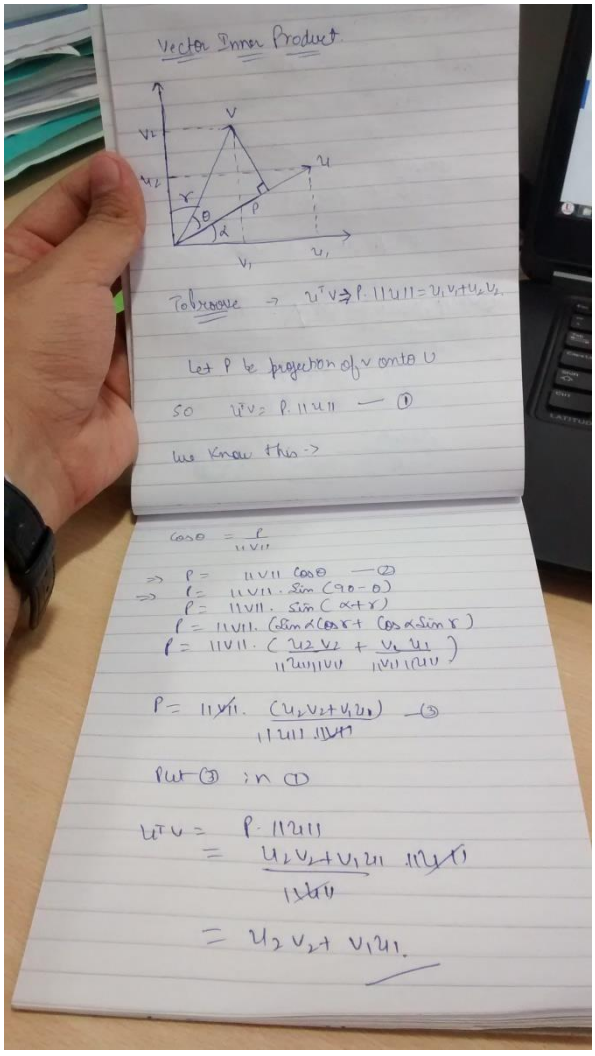
Signed

$u^T v = p \cdot \|u\| \leftarrow \quad = v^T u$

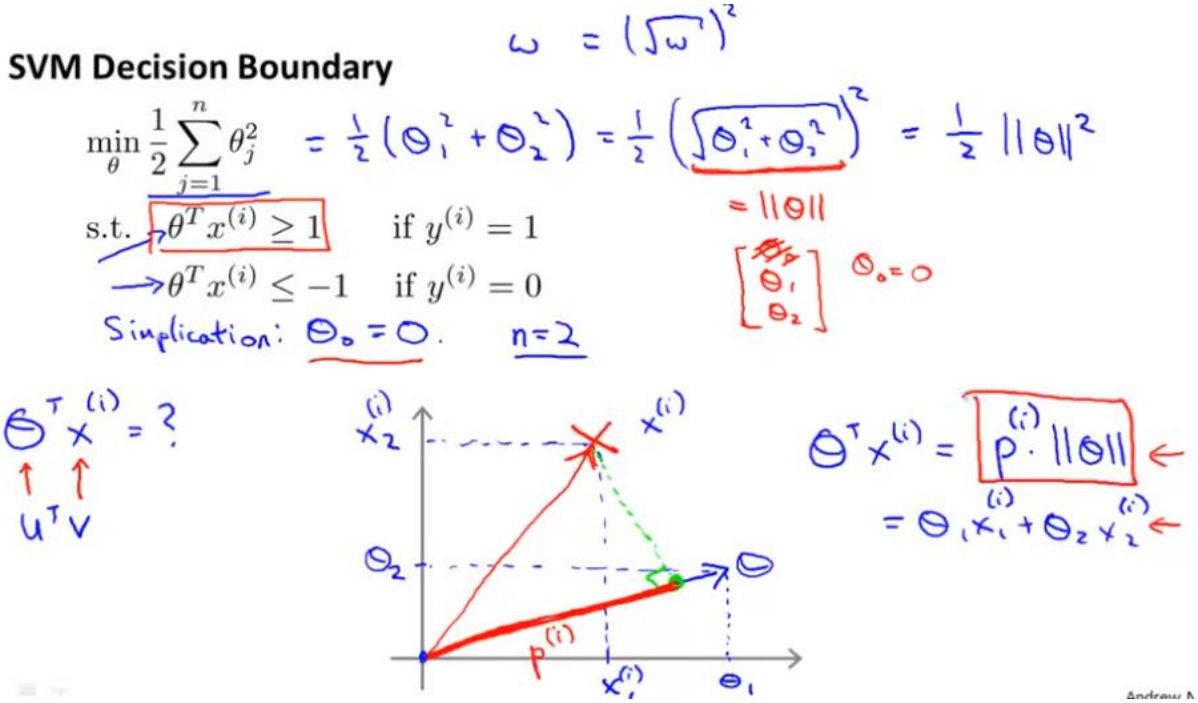
$= u_1 v_1 + u_2 v_2 \leftarrow \quad p \in \mathbb{R}$

$u^T v = p \cdot \|u\|$

$p < 0$



2) Cost function을 minimize하는 식을 다시 쓰기



→ subjected to 조건을 1)에서 증명한 식으로 다시 써보자.

3) P가 커지는 곳에 decision boundary가 그려지기 때문에 ➔ margin이 큰 곳에 결정된다.

SVM Decision Boundary

$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$$

$$\text{s.t. } p^{(i)} \cdot \|\theta\| \geq 1 \quad \text{if } y^{(i)} = 1$$
$$p^{(i)} \cdot \|\theta\| \leq -1 \quad \text{if } y^{(i)} = 1$$

where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ.

Simplification: $\theta_0 = 0$

* decision boundary와 theta가 직각인 이유

Q3) How do we know that the theta is perpendicular to the decision boundary?

(thanks to Mentor Chirag for this derivation)

We know that $\theta' * x = 0$ for any x on the boundary since the boundary is where sigmoid = 0.5.

$$\frac{1}{1+e^{-z}} = \frac{1}{2} => e^{-z} = 1 => -z = 0 => \theta' * x = 0$$

So pick two random points on the boundary a and b, then

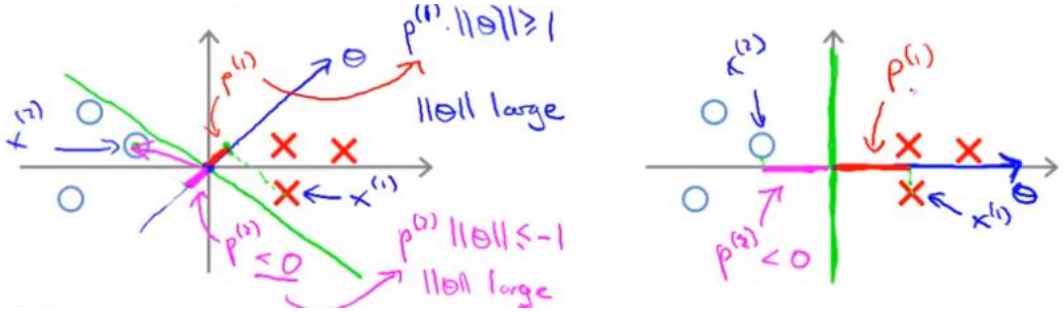
$\theta' * a = 0$ and $\theta' * b = 0$

$=> \theta' * (a - b) = 0 => \theta' \cdot (a - b) = 0$

and we know that when the dot product of two vectors is 0, the angle between them is 90 degrees. And since the vector (a-b) is on the decision boundary, θ is perpendicular to the decision boundary.

- decision boundary를 두 가지 경우로 그려서 나눠서 생각해보자.
  - decision boundary와 theta는 수직이다.
  - x에서 theta에 수직으로 내려서 projection한 벡타의 norm이 p의 값이다.

- 첫번째 경우(왼쪽)에는 p가 작아진다. 그런데 목적함수를 최소화하려면 1/2 |θ|^2를 최소화해야 하는데 그러면 p * |θ| 가 1보다 커지지 않게 될 수 있다. 따라서 제한 조건을 만족시키지 못하게 된다. 두번째 경우처럼 margin을 최대화해야 목적함수의 최소화가 이루어진다. 즉, p가 큰 곳에 decision boundary(연두색선)이 그려져야 하는 것이다.

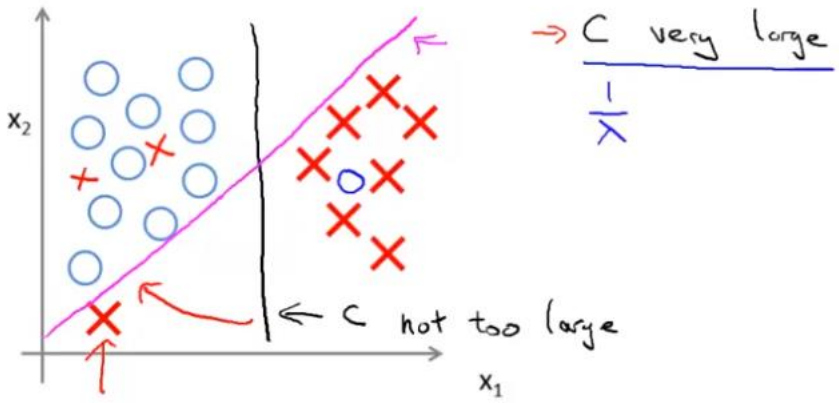


- θ0 = 0 이 아니더라도 똑같이 large margin을 찾게 될 것이다.

1.7.3. How do we use SVM? → C조절 (regularization parameter)

- fitting을 할 때, 이상값을 넣으면 안되기 때문에 C값을 낮추면 된다.

Large margin classifier in presence of outliers



→ 여기서 C가 높을 때에는 magenta line처럼 된다. C를 낮추면 검은 색 선처럼 decision boundary가 생긴다.

- 내적을 하게 되면 projection * 길이

- 목적은 theta를 구하는 것이다.
- p(i) * | theta |

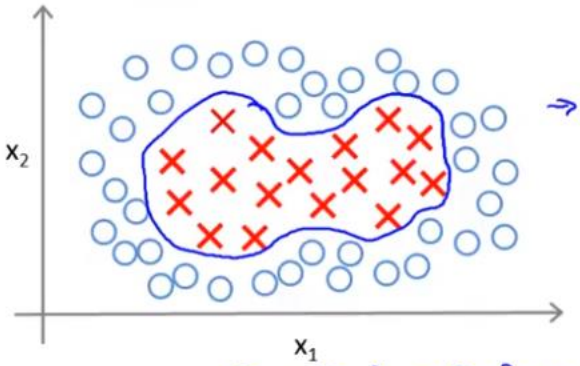
- 임의의 theta로 시작을 하고 projection을 다 시킨다. p가 클수록 좋은 것이다.
- theta를 이동시켜서 p값을 점점 크게 만든다.

Decision boundary는 package로 써라.

1.7.4. Using SVM in complex nonlinear classifiers : kernel

- 문제의 정의 : nonlinear classifier를 만들 때, 차수가 너무 많아서 computational cost가 너무 높게 된다. 따라서 여기에 SVM을 적용하면 비용을 낮출 수 있다.

Non-linear Decision Boundary



Predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \geq 0 \\ 0 & \text{otherwise.} \end{cases}$

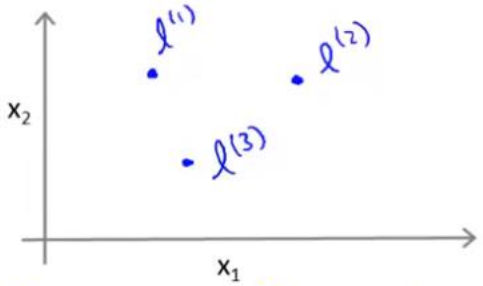
$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1 x_2, \quad f_4 = x_1^2, \quad f_5 = x_2^2, \ldots$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

- 해결 방법 : landmark와의 유사성을 f1으로 표시를 하게 되면 어떨까? similarity function은 kernel을 의미한다. kernel 중 하나가 Gaussian kernel이 있다. 다른 kernel 종류도 있다.

Kernel



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

$\|w\|$

$\|x - l^{(1)}\|^2$

Given x:

$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

$f_2 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$

$f_3 = \text{similarity}(x, l^{(3)}) = \exp(\ldots)$

kernel (Gaussian kernels)

$k(x, l^{(i)})$

- 저 식이 작동하는 방식 : 만약에 x가 landmark에 가까우면 f1은 1에 가까워진다. 그러나 landmark에서 멀어지면 f1이 0이 된다.

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

$l^{(1)} \rightarrow f_1$
$l^{(2)} \rightarrow f_2$
$l^{(3)} \rightarrow f_3$.

If x if far from $l^{(1)}$:

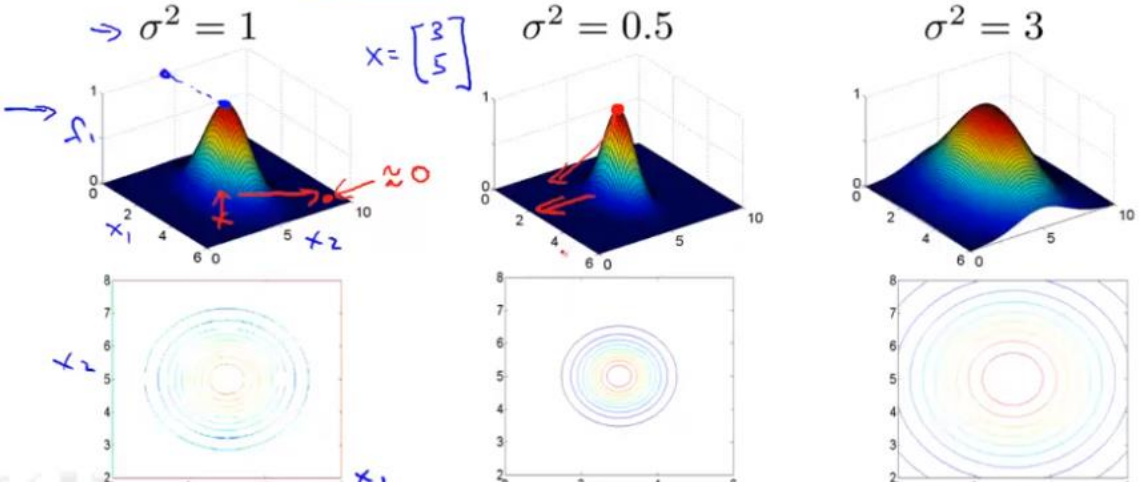
$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

- 예를 들어보자 : f1을 y축으로 하면 (x1, x2)가 3,5에 가까워질수록 1이 되므로 산 모양이 된다. 시그마 제곱이 달라지면 어떻게 될까? 만약에 1에서 0.5가 되면 y가 1에서 0이 되는 것이 더 빨라지므로 경사가 급격해지고, 만약에 시그마 제곱이 3이 되면 y가 1에서 0이 되는 것이 더 느려지므로(분모가 커졌으므로 음의 절대값이 더 느리게 커진다) 경사가 완만해진다.

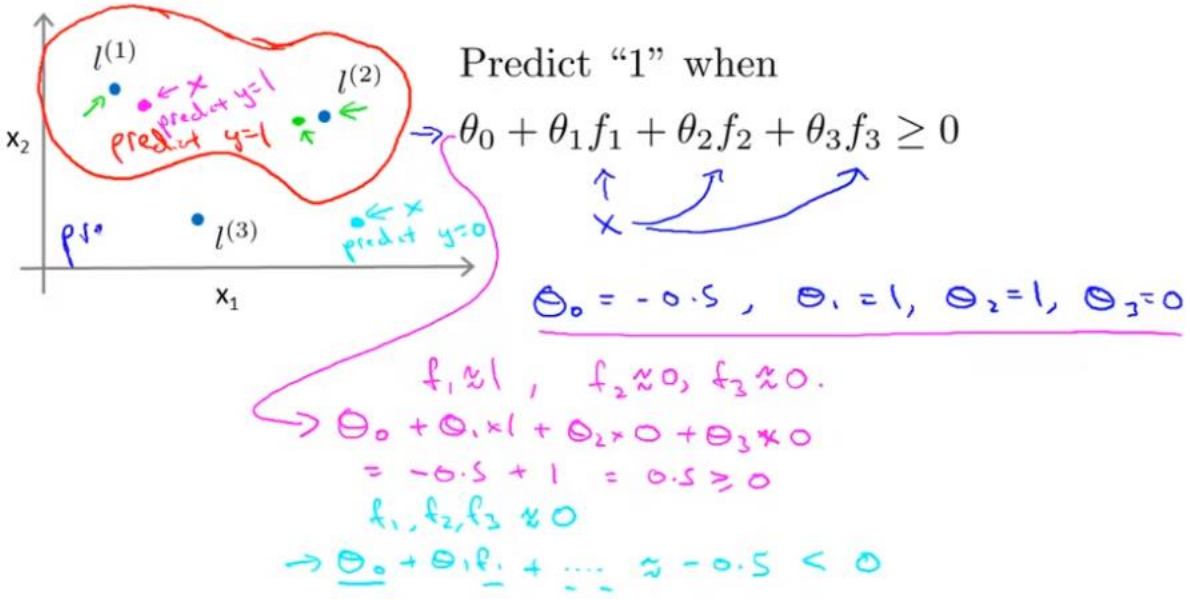
Example:

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

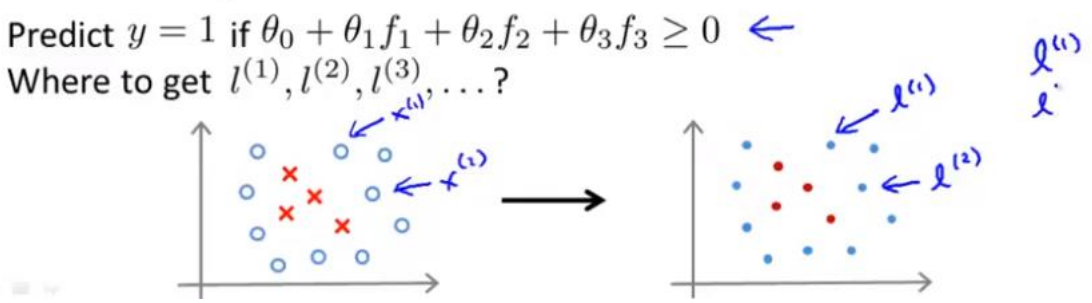
$\sigma^2 = 1$ $\sigma^2 = 0.5$ $\sigma^2 = 3$



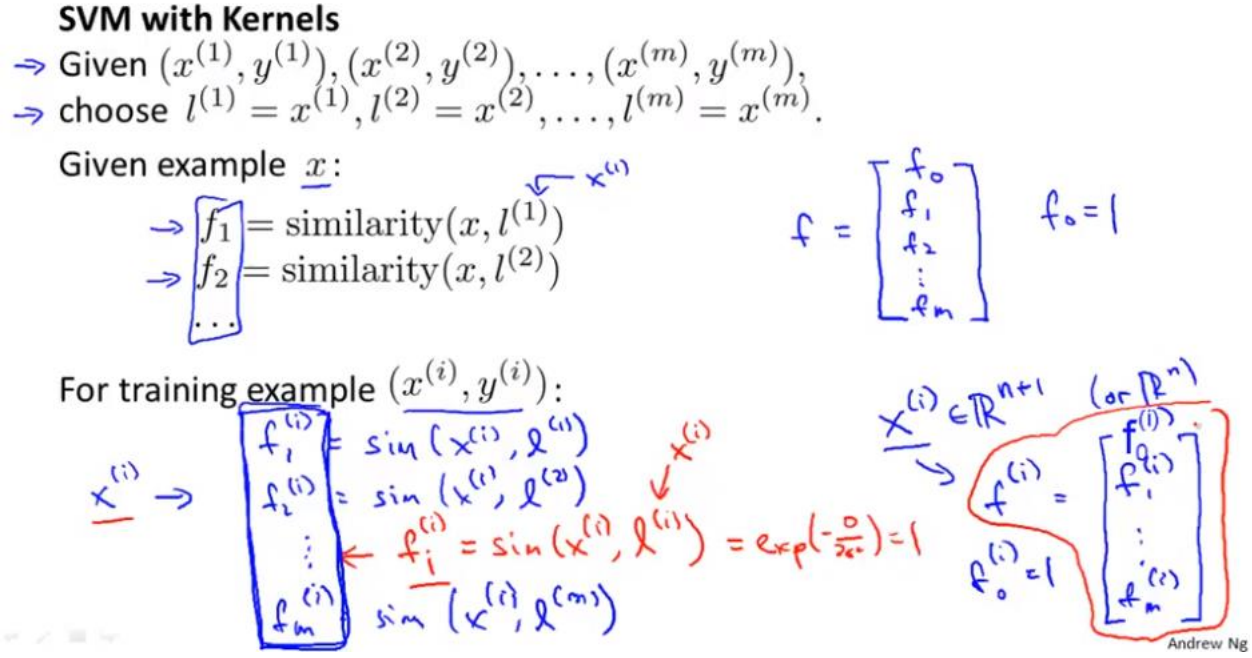
- 적용 : 만약에 핑크색 점의 data가 있다고 해보자. 이는 landmark 1과 가깝기 때문에 f1=1, f2=0, f3=0일 것이다. 따라서 이를 parameter를 넣고서 계산해보면 0보다 크므로 y=1을 predict하게 된다.



- kernel의 설계 :
  - 어떻게 landmark를 설정할 것인가? : m개의 training set을 그래도 landmark로 한다.



  - 벡터로 만들어버린다.



  - min 목적식을 만들어보자.

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

→ Predict "y=1" if $\theta^T f \geq 0$

$\theta \in \mathbb{R}^{n+1}$

$\theta_0 f_0 + \theta_1 f_1 + \cdots + \theta_m f_m$

$n = m$

Training:

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

$m = m$

$\theta^T f^{(i)}$

$\rightarrow \theta_0$

- regularization term이 조금은 변형된다.(computationally expensive한 것을 방지하기 위해)

$\sum_{j} \theta_j^2 = \theta^T \theta \leftarrow \theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$ (ignore θ_0)

$\leftarrow \|\theta\|^2$

$\theta^T M \theta$

- parameter를 어떻게 결정할 것인가?
  - C의 선택과 시그마 제곱의 선택

SVM parameters:

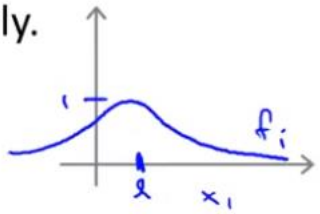
C ($= \frac{1}{\lambda}$). → Large C: Lower bias, high variance. (small λ)

→ Small C: Higher bias, low variance. (large λ)

σ^2 Large σ^2: Features f_i vary more smoothly.

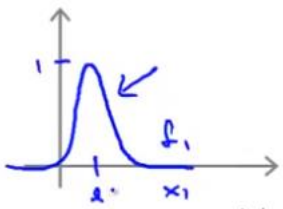
→ Higher bias, lower variance.

$\exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



Suppose you train an SVM and find it overfits your training data. Which of these would be a reasonable next step? Check all that apply.

- [ ] Increase C

  Un-selected is correct

- [x] Decrease C

  Correct

- [x] Increase σ^2

→ overfitting이 일어나게 되면 C를 줄이고, 시그마 제곱을 높게 하면 어느 정도 해결된다.

1.7.5. 실제 SVM의 활용 방법(Using an SVM)

- 실제로 이를 코딩하려고 하지 말고, library를 써라.
- linear kernel(kernel을 사용하지 않는 경우) : feature가 많고, dataset 개수(m)는 적은 경우
- kernel : feature 개수(n)가 작고, data set 개수(m)은 큰 경우가 좋다.

Use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters θ.

Need to specify:

→ Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")

Predict "y = 1" if $\theta^T x \geq 0$

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

→ n large, m small $x \in \mathbb{R}^{n+1}$

Gaussian kernel:

$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$.

Need to choose σ^2.

$x \in \mathbb{R}^n$, n small and/or m large

- Gaussian kernel을 사용하는 방법
  1) 함수를 직접 넣어줘야 하는 경우가 많다.
  2) feature scaling을 해주는 게 좋다.

Kernel (similarity) functions:

$x^{(i)}$, $l^{(j)} = x^{(j)}$

```
function f = kernel(x1,x2)
    f = exp( -|| x1 - x2 ||^2 / (2*sigma^2) )
return
```

f_i

$x \rightarrow \begin{matrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{matrix}$

→ Note: Do perform feature scaling before using the Gaussian kernel.

$x \in \mathbb{R}^n$

$||x - l||^2$

$v = x - l$

$||v||^2 = v_1^2 + v_2^2 + \cdots + v_n^2$

$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$

1000 feet² 1-5 bedrooms

3) kernel식이 Mercer's Theorem을 만족해야 한다.

Other choices of kernel

Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels.

→ (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

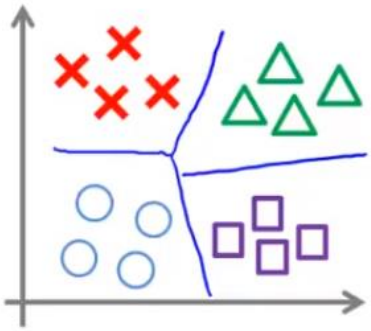
Many off-the-shelf kernels available:

- Polynomial kernel: $k(x, l) = (x^T l)^2$, $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$, $(x^T l + \text{constant})^{\text{degree}}$

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

4) multi-class classification 사용법

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

→ Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

Pick class i with largest $(\theta^{(i)})^T x$

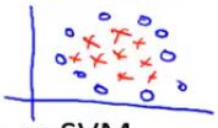
5) SVM vs logistic regression

- kernel SVM은 feature가 작고 dataset이 보통일 때 많이 쓰인다.
- feature가 많고 dataset이 적거나, feature가 적고 dataset이 엄청 많으면 linear kernel 혹은 logistic regression을 쓴다.
- neural network도 적용이 가능하다. 그런데 neural network가 training할 때 더 느릴 수 있다.
- SVM은 convex function minimization이기 때문에 global minimum을 찾아준다. local optima 걱정을 할 필요는 없다.

Logistic regression vs. SVMs

n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

- If n is large (relative to m): (e.g. $n \geq m$, $n = 10{,}000$, $m = 10 \cdots 1000$)
  - Use logistic regression, or SVM without a kernel ("linear kernel")
- If n is small, m is intermediate: ($n = 1 - 1000$, $m = 10 - 10{,}000$)
  - Use SVM with Gaussian kernel
- If n is small, m is large: ($n = 1 - 1000$, $m = 50{,}000+$)
  - Create/add more features, then use logistic regression or SVM without a kernel



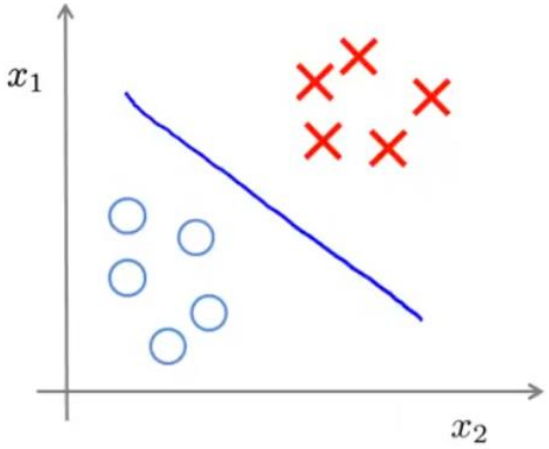
- Neural network likely to work well for most of these settings, but may be slower to train.

Week 8.

2. Unsupervised learning ← unlabeled data set

- 복습 : supervised learning

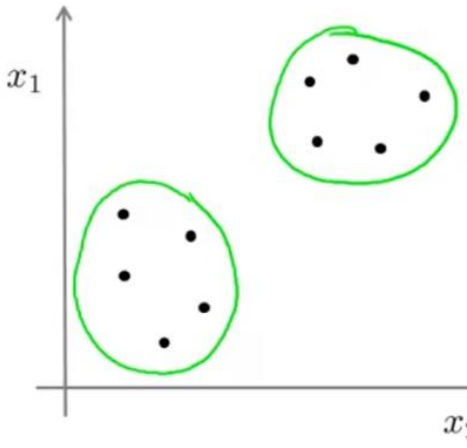
Supervised learning



Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(m)}, y^{(m)})\}$

- 차이점 : algorithm으로 find some structure for us!

Unsupervised learning

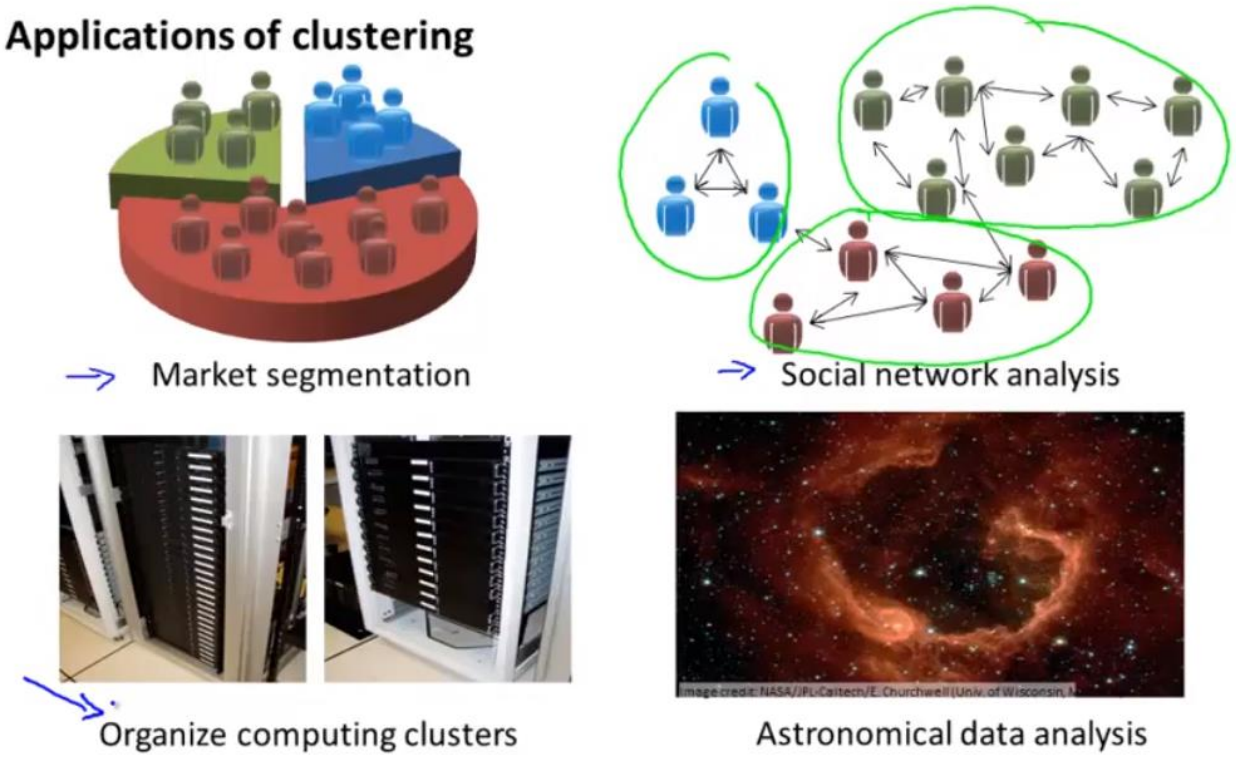


Clustering algorithm

Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$

→ clustering하는 방법도 unsupervised 중 하나이다.

- 예



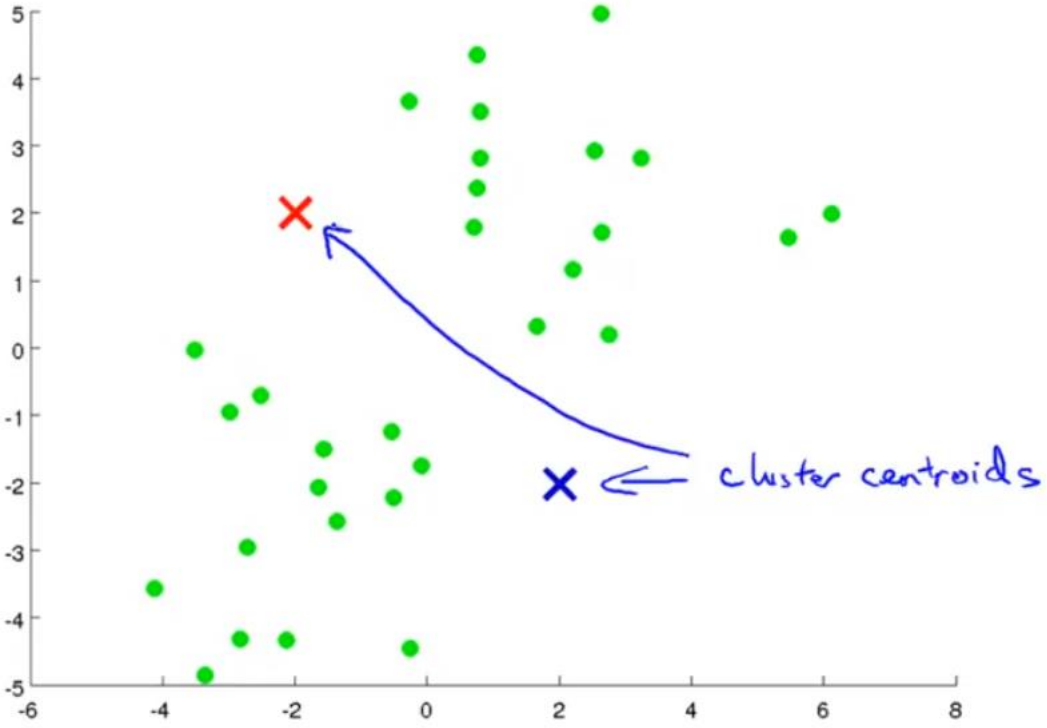
2.1. Unsupervised learning 종류

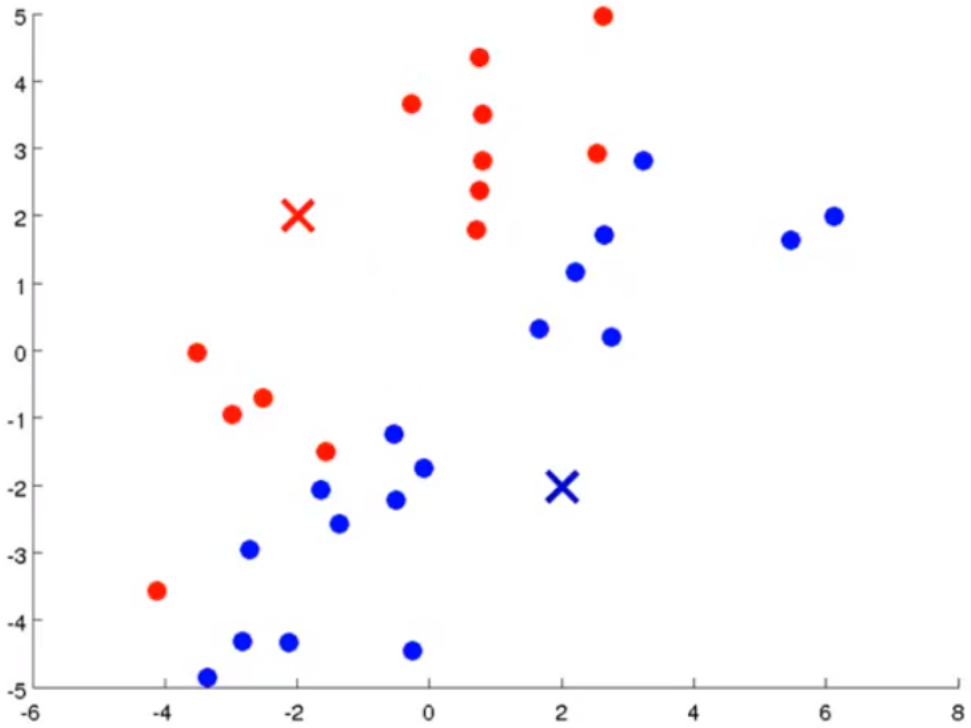
2.1.1. Clustering algorithm : K-means algorithm (the most popular)

(1) K-means Algorithm visualization : 예시로 보는 warm-up

(1단계) cluster assignment step

→ 거리 더 가까운 것을 각각 다른 색깔로

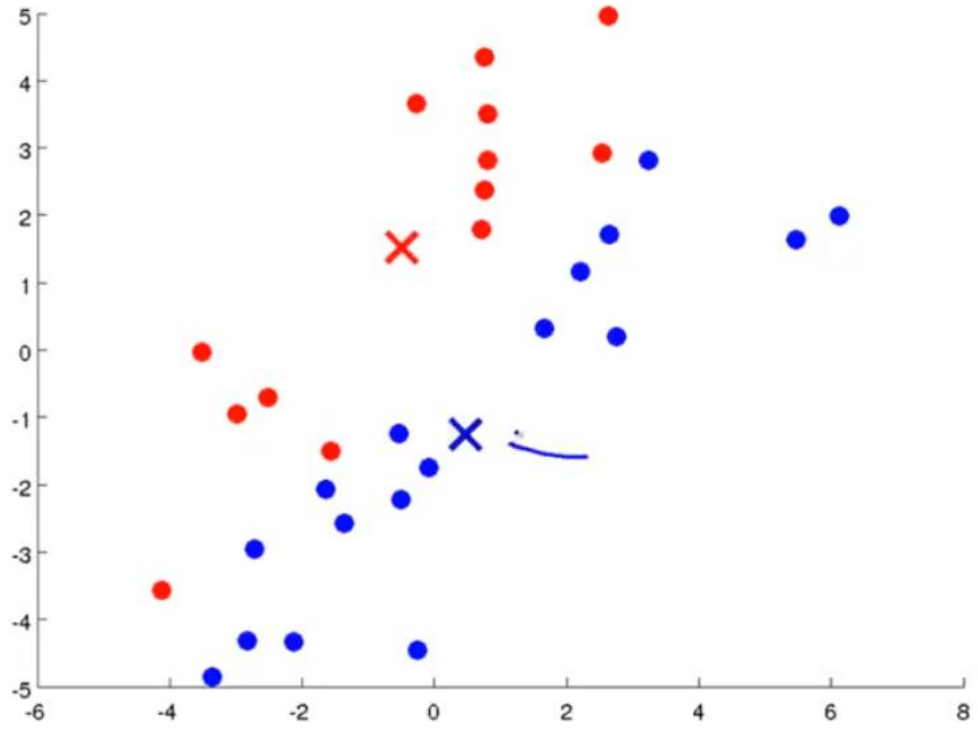




-

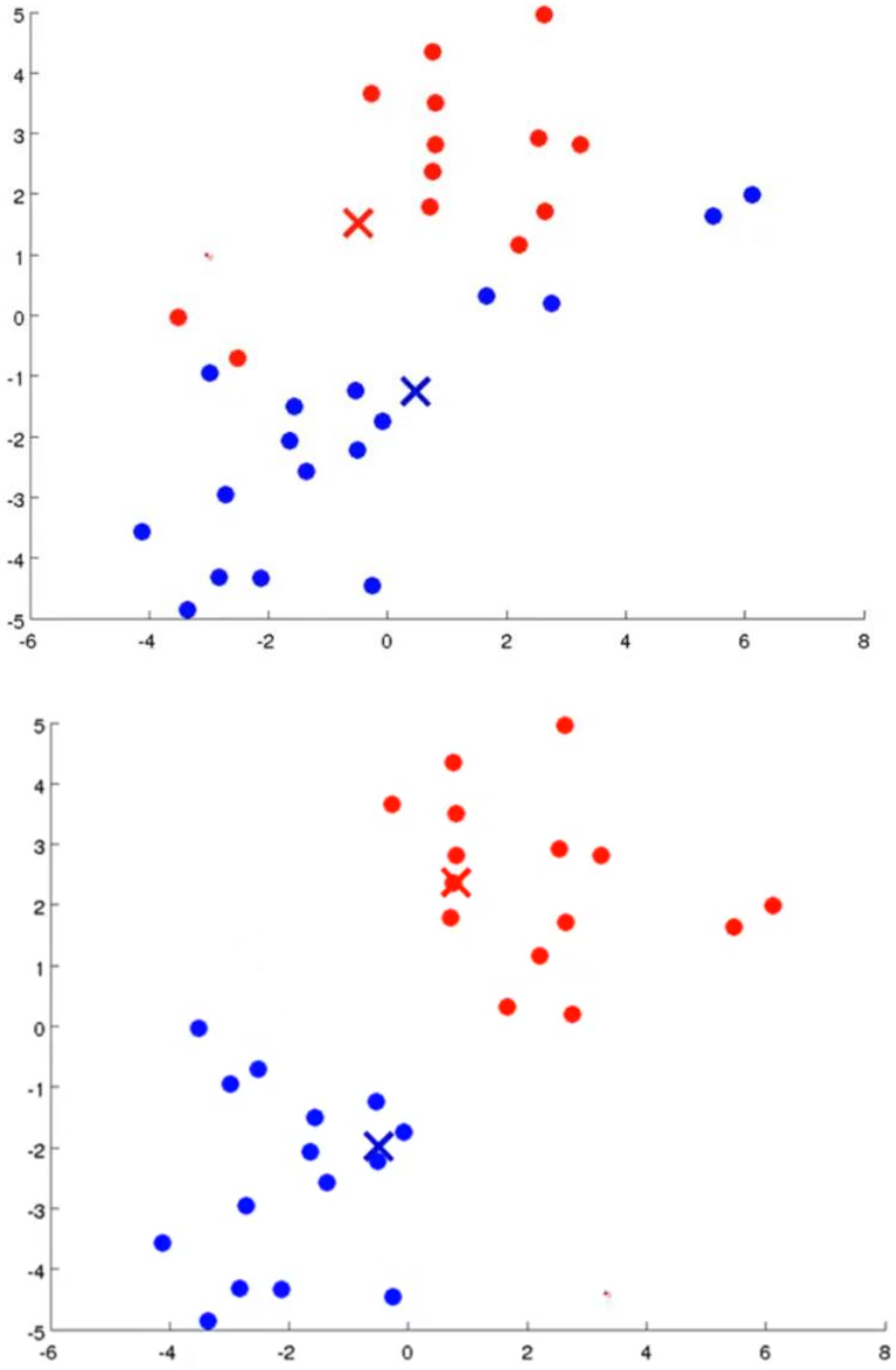
(2단계) move centroids step

- 모든 파란 점을 보고 평균을 계산하여 그 자리로 파란 x를 이동시킨다. 빨간 점도 x, y축을 모두 더하여 n으로 나누고 그 쪽으로 빨간 x를 이동시킨다.

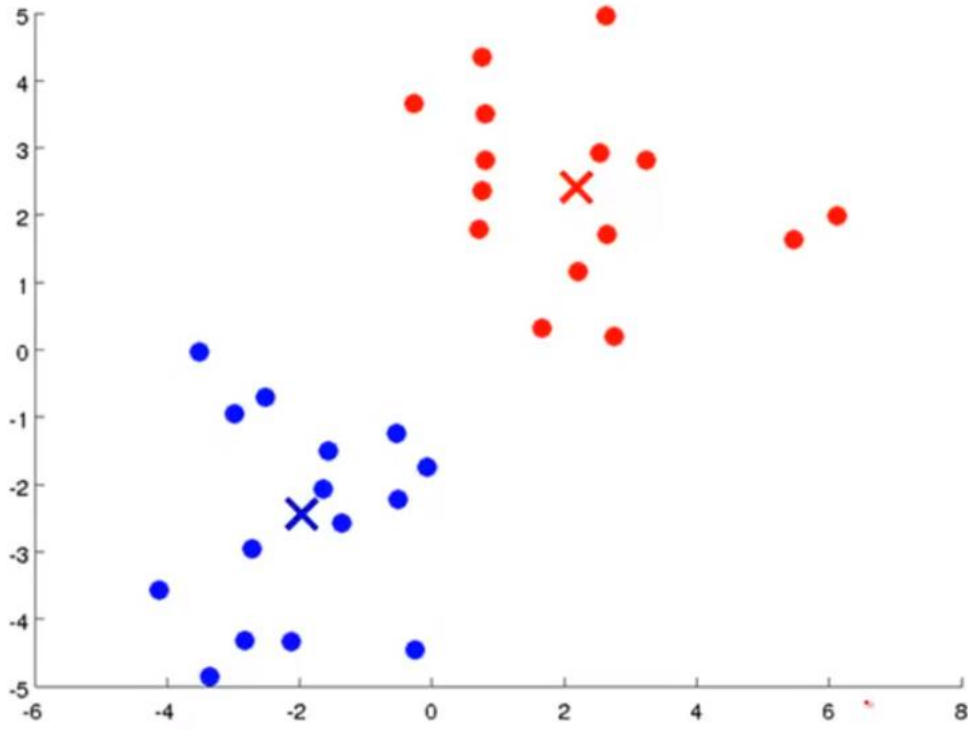


(다시 1단계)

cluster centroid에 거리가 가까운 것을 빨간, 파란색으로 표시한다.



(다시 2단계) move centroid step



→ 완성!!

(2) K-mean algorithm

- 몇 개의 cluster를 설정할 것인가? → k를 정해 (k를 automatically 정하는 알고리즘도 있다.)

Step1. Input : K를 정하는 중요한 단계!

K-means algorithm

Input:

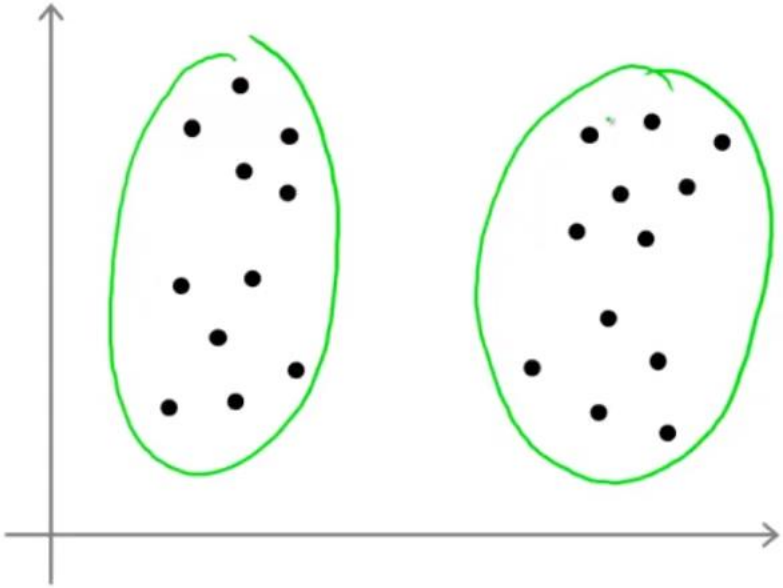
- K (number of clusters) ←
- Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ ←

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

→ label이 없어졌으므로 n개 항으로 이루어진 vector (그 전에는 n+1이었다.)

- capital K (cluster의 개수)를 정하는 방법

(방법 1) visualization을 보고 manually 하는 게 많이 사용된다. → ambiguous할 수 있다.

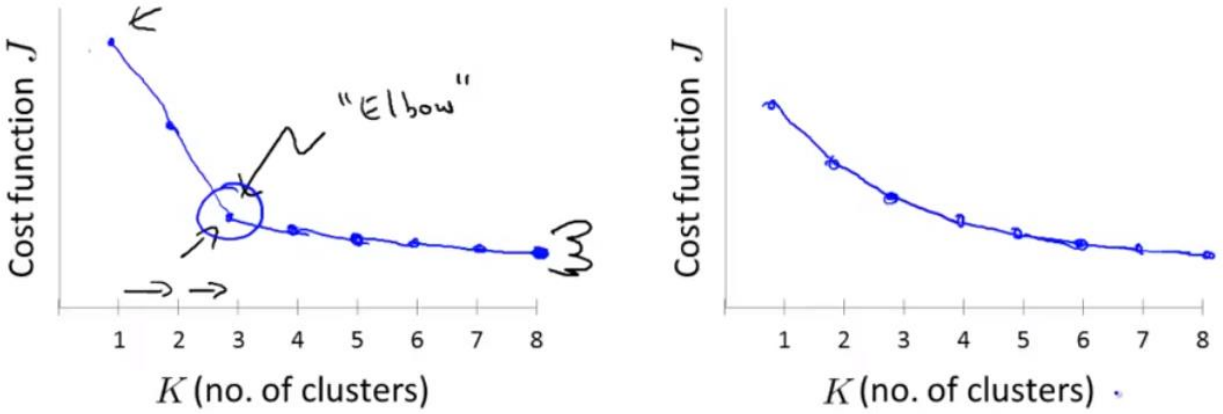


(방법 2) Elbow method도 있다. J cost function이랑 number of cluster를 plotting하는 방법도 있다.

: elbow처럼 팍 꺽이는 점에서 K를 정한다. 가파르게 cost가 줄어드는 지점에서! 그러나, 실제로는 팍 꺾이는 지점이 나타나지 않는 경우가 흔하기 때문에 elbow method가 사용하기가 애매하다.

Choosing the value of K

Elbow method:

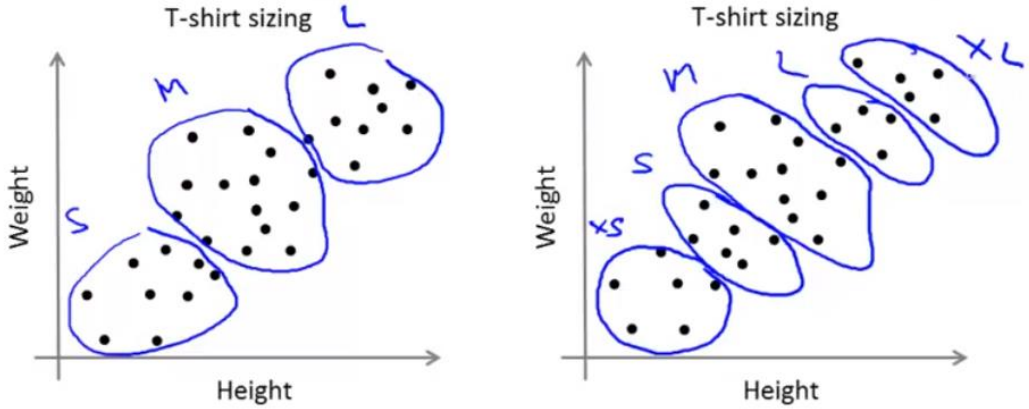


(방법 3) downstream(later) purpose에 부합하는 K를 고른다. ex) T-shirt segmentation

Choosing the value of K

Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

E.g.



→ 어떤 T-shirt size cluster로 해야지 좋을까? 소비자에게 딱 맞는 shirt를 만드는 게 중요한가, 아니면 규모의 경제를 실현하도록 size option을 줄이는 게 좋을까?

- image compression을 할 때에도, file size를 줄이는 게 중요한가, 아니면 image가 잘 보이는 게 중요한가.

Step2. 다음 algorithm

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

(Cluster assignment step)

for $i = 1$ to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

for $k = 1$ to K

μ_k := average (mean) of points assigned to cluster k

}

(2-1) How to randomly initialize K cluster centroid → to avoid local optima

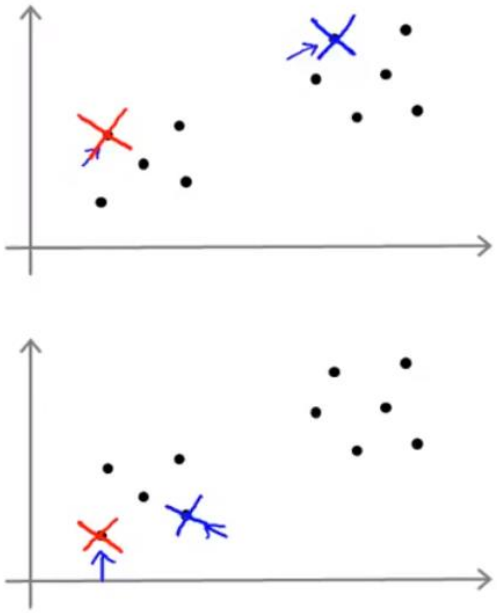
- x data set 중에서 random하게 정하는 방법을 강력 추천한다.

Random initialization

Should have $K < m$

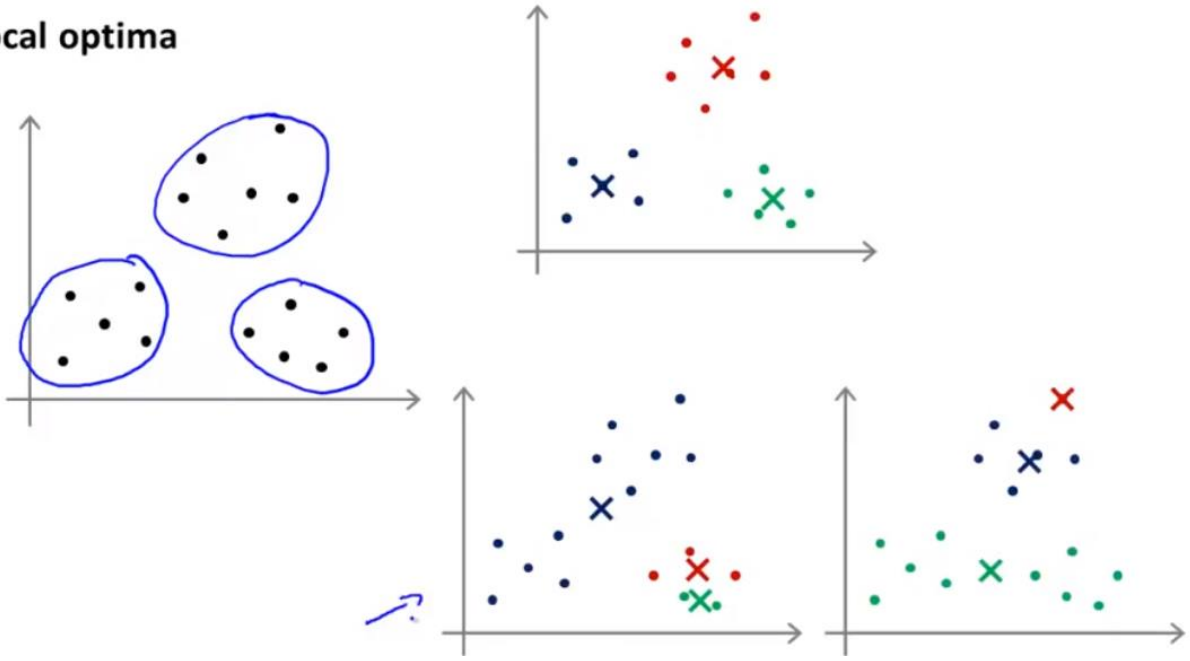
Randomly pick K training examples.

Set $\mu_1, \ldots, \mu_K$ equal to these K examples.



- local optima of J (cost function)의 함정과 해결

Local optima



→ multiple random initialization

→ k-means를 많이 해서 (100번 이상) 가장 best의 clusterization을 고른다. i는 iteration의 i이다. 따라서 몇 번을 찍을 것인가를 의미한다.

Random initialization

For i = 1 to 100 { 50 - 1000

→ Randomly initialize K-means.
Run K-means. Get $c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K$.
Compute cost function (distortion)
→ $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$
}

Pick clustering that gave lowest cost $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$

k = 2 - 10

→ k (cluster의 개수)가 2~10이면 random initialization이 효과가 좋으나, cluster가 100개 이면 사실상 random initialization이 큰 개선 효과가 없다.

(2-2) cluster assignment step : data set x를 i = 1 to m까지 min 거리하는 μk의 k로 indexing을 한다.
(2-3) move centroid step :

K-means algorithm

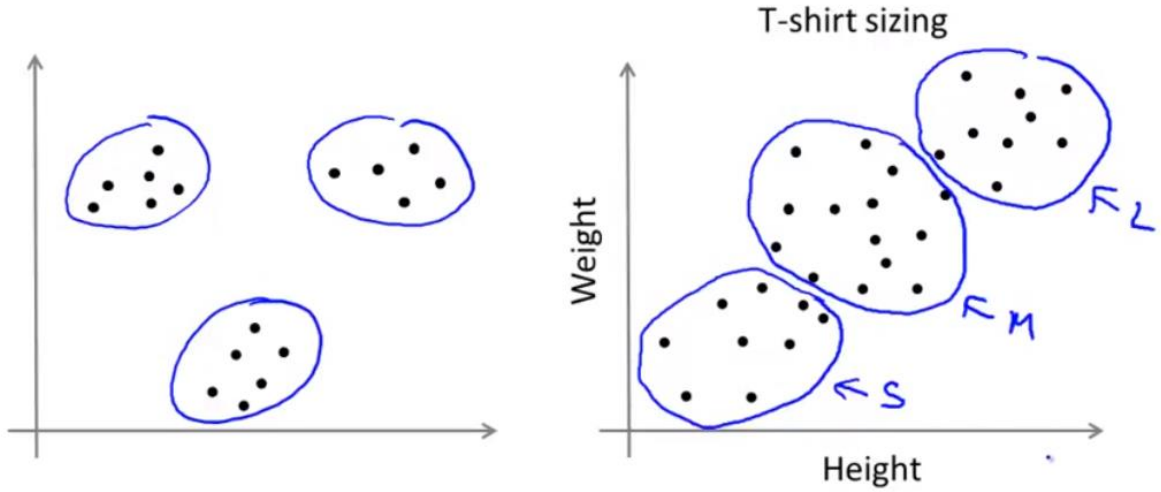
Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$
Repeat {
Cluster assignment step
for i = 1 to m
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$
$\min_k \|x^{(i)} - \mu_k\|^2$ → $c^{(i)}$
Move centroid
for k = 1 to K
→ μ_k := average (mean) of points assigned to cluster k
$x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)}$ → $c^{(1)}=2$, $c^{(5)}=2$, $c^{(6)}=2$, $c^{(10)}=2$
$\mu_2 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right] \in \mathbb{R}^n$
}

→ cf) 만약에 k번째 cluster에 아무 data set도 배정되지 않으면 eliminate시키거나, 다시 random하게 μk를 설정해주는 방법이 있다. 그러나 전자가 더 흔하다.

- non well separated cluster도 잘된다는 점.
  - T-shirt sizing을 하려는데 clustering이 잘 안 되는 경우, S, M, L을 잘 구분할 수가 없다. 그래도 k-mean을 하면 나름 구분을 할 수 있다. 좋네.

K-means for non-separated clusters

S, M, L



2.1.3. optimization objective

- 목적 : debugging을 할 수 있다. 더 중요하게는 local optimum을 피할 수 있다.
- 변수 :

→ $c^{(i)}$ = index of cluster (1,2,...,K) to which example $x^{(i)}$ is currently assigned

→ μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

K $k \in \{1,2,\ldots,K\}$

$x^{(i)} \rightarrow 5$ $c^{(i)} = 5$ $\mu_{c^{(i)}} = \mu_5$

- objective

Optimization objective:

$$\rightarrow J(c^{(1)},\ldots,c^{(m)},\mu_1,\ldots,\mu_K) = \frac{1}{m}\sum_{i=1}^{m}||x^{(i)} - \mu_{c^{(i)}}||^2$$

$$\min_{\substack{c^{(1)},\ldots,c^{(m)},\\ \mu_1,\ldots,\mu_K}} J(c^{(1)},\ldots,c^{(m)},\mu_1,\ldots,\mu_K)$$

→ x와 x가 배정된 cluster의 centroid와의 거리 차이를 minimize하면 된다.

- 아까의 algorithm에서 위의 minimize를 하는 것이다. 즉, cluster assignment step에서는 c(i)에 대하여 J cost function을 minimize하고, move centroid step에서는 μ_k에 대하여 cost function을 minimize 하는 것이다. Cost function은 distortion function이라고도 한다.

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

Repeat {

for i = 1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

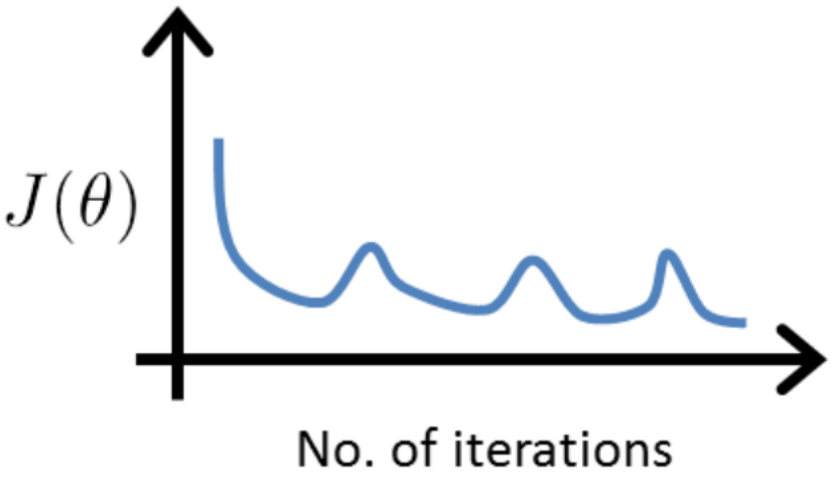
for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

}

- 만약에 이러면?

Suppose you have implemented k-means and to check that it is running correctly, you plot the cost function $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k)$ as a function of the number of iterations. Your plot looks like this:



It is not possible for the cost function to sometimes increase. There must be a bug in the code.

Correct

→

Unsupervised learning 종류-2

2.2. Dimensionality reduction

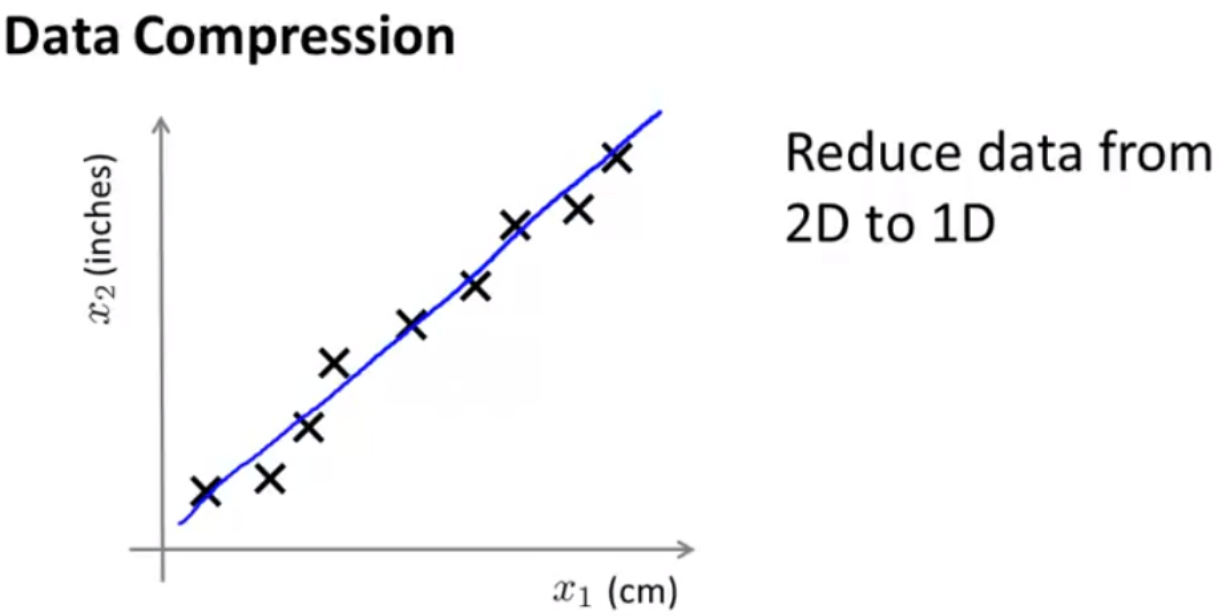
- 목적 :

(1) data compression → speed up algorithms / reduce data storing space

(2) Data visualization

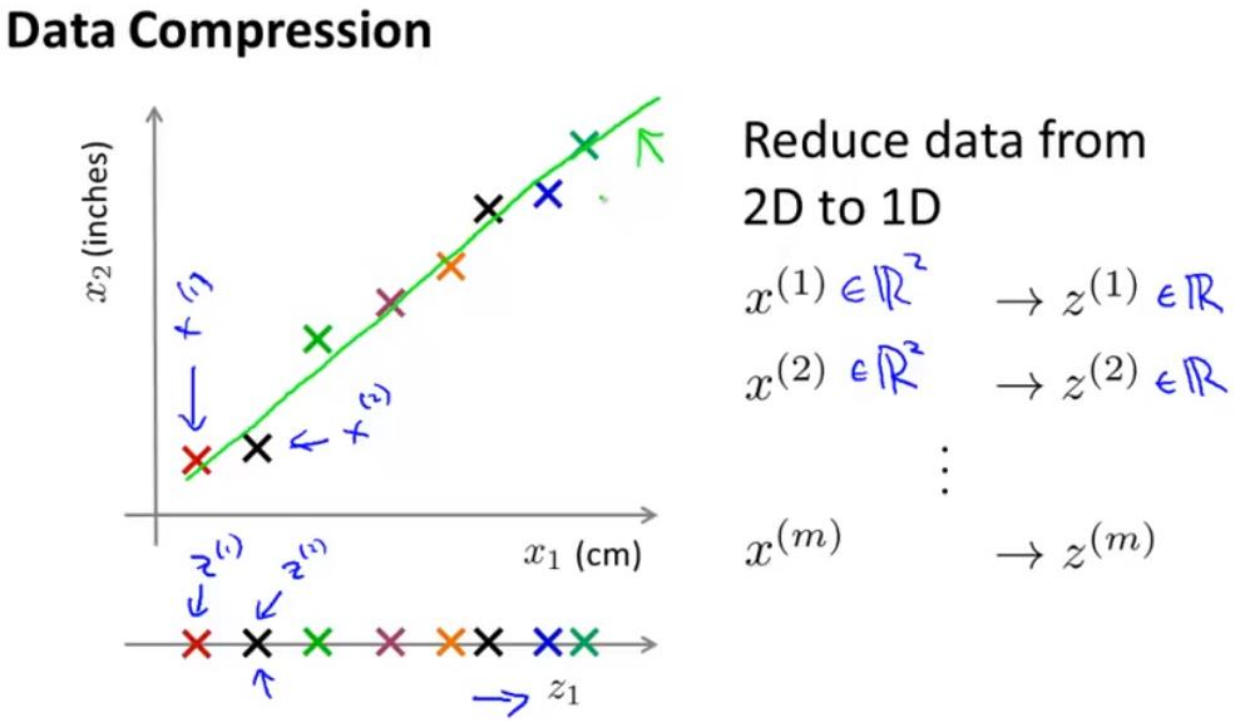
- 실제 알고리즘 : Principal Component Analysis

2.2.1. Data compression 원리 (reduce the dimension)



→ round off error가 생겨서 왔다갔다 한다.

- 2개의 dimension을 1개의 dimension으로 줄이는 예 : new feature z1을 도입하여 x1, x2를 대변할 수 있게 된다.



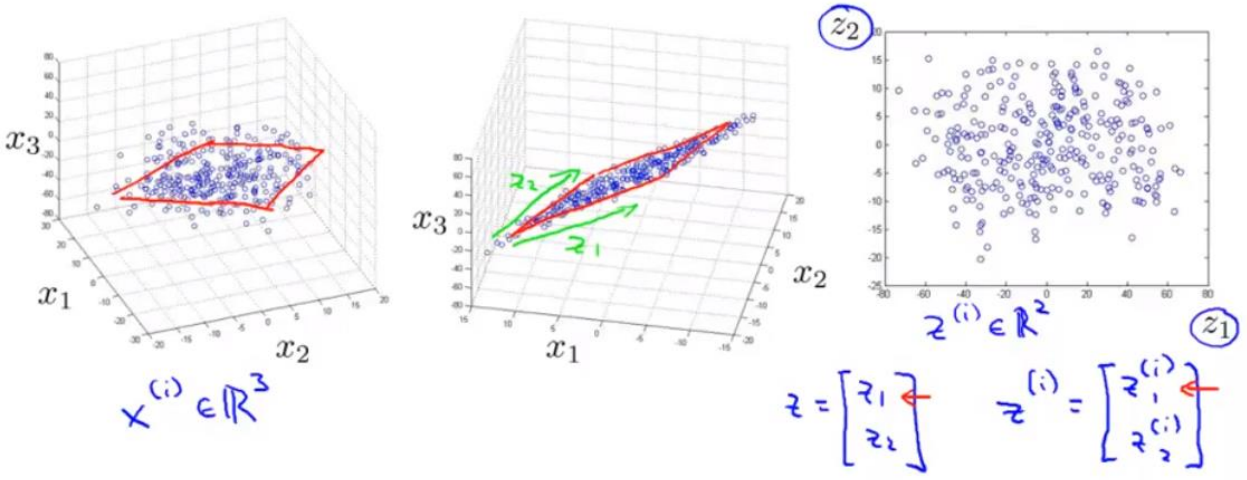
- 3D에서 2D로 reduce하는 예

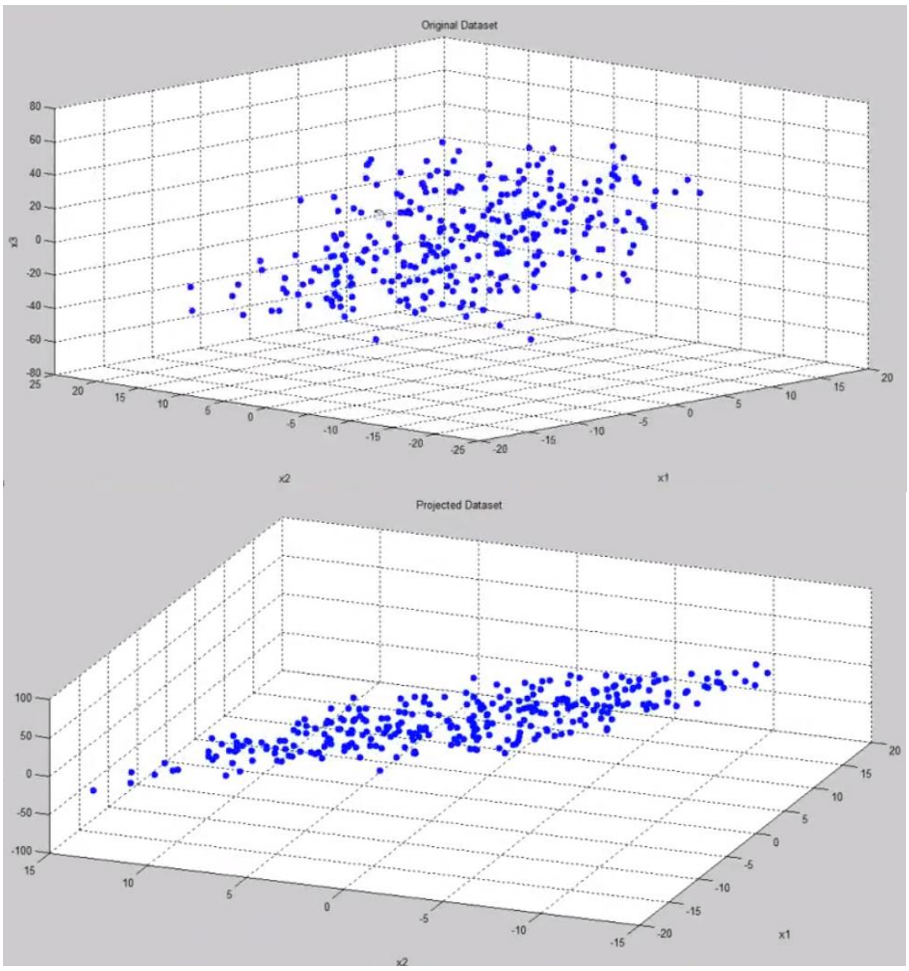
  : project x3 to the x1-x2 plane

Data Compression

10000 → 1000

Reduce data from 3D to 2D





- 차원을 줄이는 것이다! data set의 개수를 줄이는 것이 아니라!

Suppose we apply dimensionality reduction to a dataset of m examples $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, where $x^{(i)} \in \mathbb{R}^n$. As a result of this, we will get out:

- A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(k)}\}$ of k examples where $k \leq n$.
- A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(k)}\}$ of k examples where $k > n$.
- (●) A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ of m examples where $z^{(i)} \in \mathbb{R}^k$ for some value of k and $k \leq n$.
- A lower dimensional dataset $\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\}$ of m examples where $z^{(i)} \in \mathbb{R}^k$ for some value of k and $k > n$.

2.2.2. Data visualization ← dimension reduction의 목적-2

- 원래 데이터 (x vector의 차원은 50)

Data Visualization

$x \in \mathbb{R}^{50}$ $x^{(i)} \in \mathbb{R}^{50}$

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	

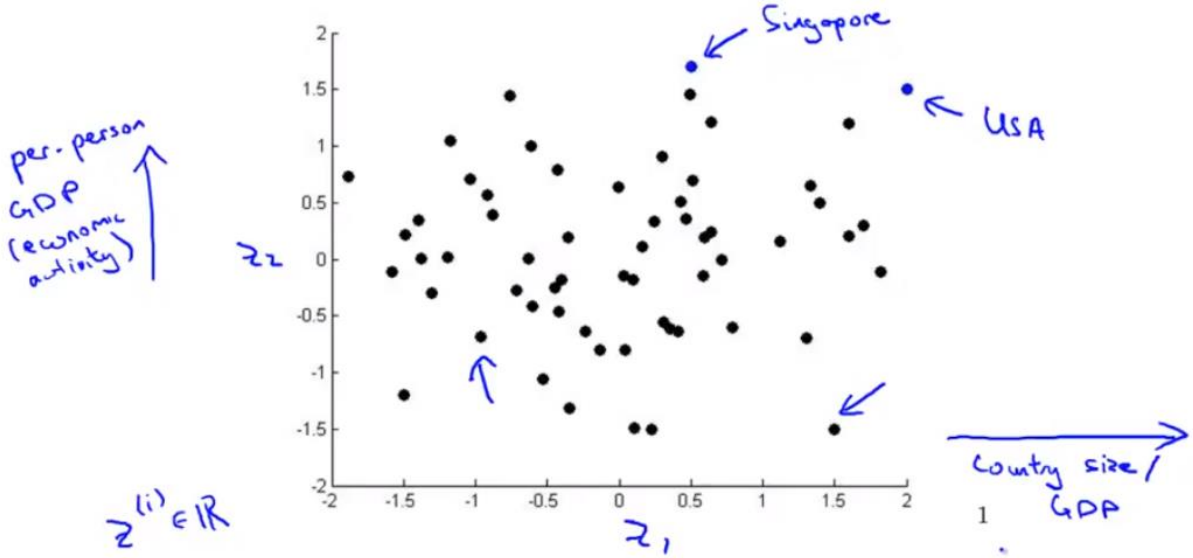
- 가공한 data (z vector의 차원은 2)

Data Visualization

$z^{(i)} \in \mathbb{R}^2$

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5
Singapore	0.5	1.7
USA	2	1.5
...	...	...

Reduce data from 50D to 2D



→ 질문!! 그런데 어떻게 2개로 줄이지????? PCA 대박!!!

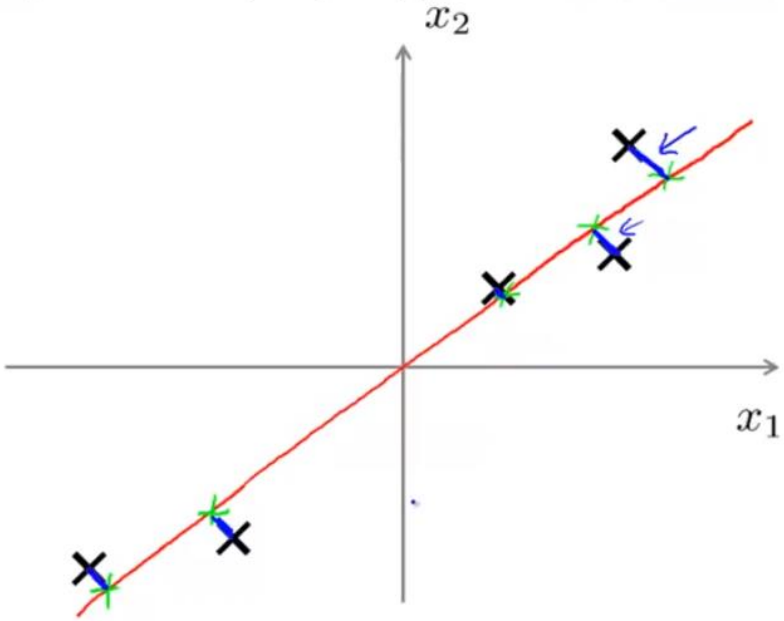
2.3. (중요) Principle component analysis (PCA)

2.3.1. 문제정의 : PCA Problem formulation

➔ find lower dimensional surface by projecting data

- 2D 예시 : 2차원 데이터가 있으면, lower dimensional surface(line) = projection surface / projection line을 찾아내서 dimension을 줄인다. 그러려면 projection error (square projection error)를 minimize한다. 우선은, 이를 하기 전에 feature scaling / mean normalization을 한다.(평균을 1, range를 -3, 3 사이로 맞춘다.

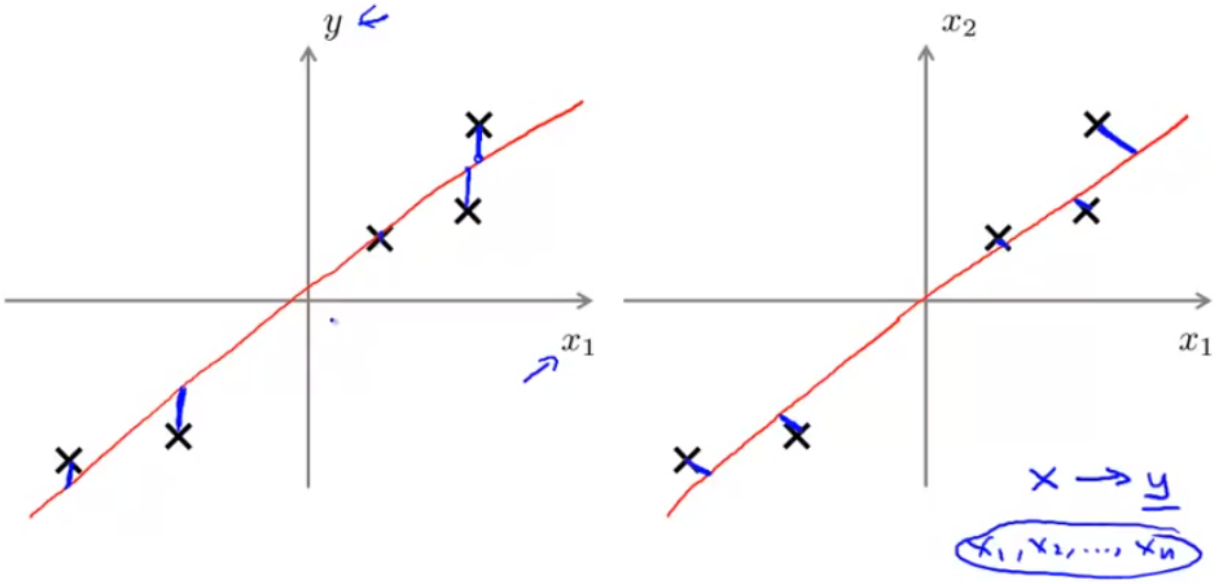
Principal Component Analysis (PCA) problem formulation



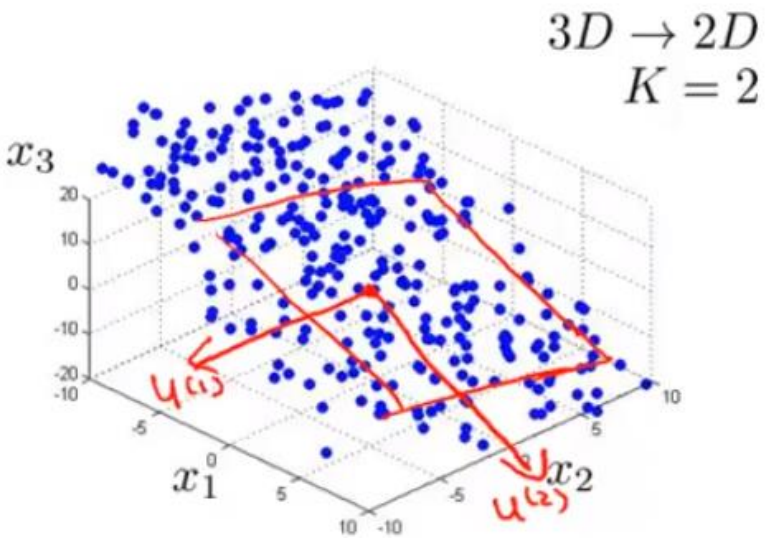
Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

cf) linear regression과의 차이점 : linear regression은 y축으로 오차(y를 기준으로 차이)를 구하지만, PCA는 projection을 한다. 즉, 특별한 y가 없다. 이것이 바로 unsupervised와의 가장 큰 차이점이다.

PCA is not linear regression



→ normalization을 했기 때문에 PCA는 (0.0)을 지난다. mean이 0이기 때문이다.



Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

→ k개의 vector를 구한다.

2.3.2. 해결 알고리즘 : PCA algorithm

(1) Data preprocessing step : mean normalization하고 feature scaling한다.

Data preprocessing

Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

Preprocessing (feature scaling/mean normalization):

$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

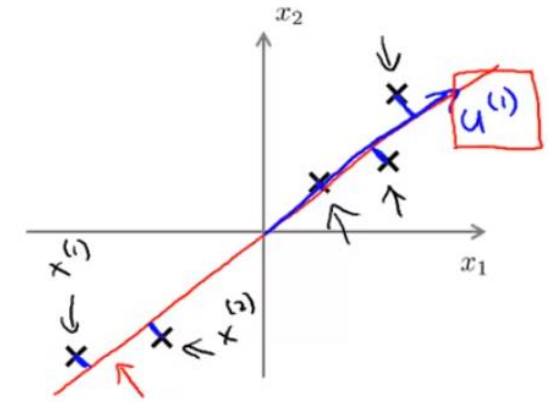
If different features on different scales (e.g., x_1 = size of house, x_2 = number of bedrooms), scale features to have comparable range of values.

$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$

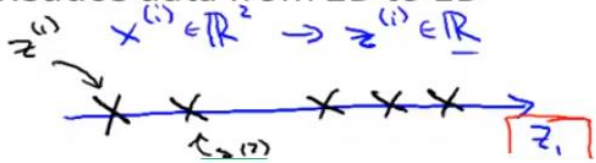
(2) reduce dimension

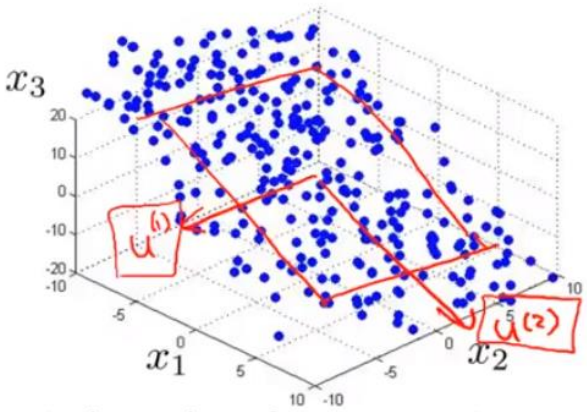
- 먼저 : u(1) vector를 찾아야지. 3D data면 u(1), u(2) vector를 찾고
- 다음 : Z1을 찾아야지

Principal Component Analysis (PCA) algorithm



Reduce data from 2D to 1D





Reduce data from 3D to 2D

$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$

$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

- 수학적 proof : Best low rank approximation을 해준다.

Solving the linear least-squares problem with the SVD

- $A = U\Sigma V^T$ where $U \in \mathbb{R}^{m\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$, $V \in \mathbb{R}^{n\times r}$.
- Assume A has rank n (i.e., $r = n$)
- Solve $Ax \approx b$:

$$
\begin{aligned}
x &= (A^TA)^{-1}A^Tb \\
&= ((U\Sigma V^T)^T(U\Sigma V^T))^{-1}(U\Sigma V^T)^Tb \\
&= (V\Sigma^TU^TU\Sigma V^T)^{-1}V\Sigma^TU^Tb \\
&= (V\Sigma\Sigma V^T)^{-1}V\Sigma U^Tb \\
&= ((V^T)^{-1}(\Sigma\Sigma)^{-1}V^{-1})V\Sigma U^Tb \\
&= V^T\Sigma^{-1}\Sigma^{-1}\Sigma U^Tb \\
&= V^T\Sigma^{-1}U^Tb
\end{aligned}
$$

- 알고리즘 : n dimension을 k dimension으로 줄이고 싶을 때

<PCA에서 K를 설정하는 방법>

- PCA 분석에서의 K 설정 (number of principal components)
- rule of thumb

Choosing k (number of principal components)

Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$

Typically, choose k to be smallest value so that

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \qquad (1\%)$$

"99% of variance is retained"

→ 이 표현의 뜻은 1%(0.01) 보다 비율이 작다는 것이다. 95% (0.05), 90% (0.1) of variance retained도 용인된다.

- 알고리즘
  - 비효율적인 알고리즘 : k=1부터 k=2로 놀여가면서 ratio가 0.01보다 작은지 본다. 가장 마지막 k를 채택하는 방법이 있다.

Algorithm:

Try PCA with $k = 1$

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$

Check if

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01?$$

$k = 17$

- 한방에 계산하는 알고리즘 : [U, S, V] = svd (Sigma)

질문!!! 왜 같은 거지???

Choosing k (number of principal components)

Algorithm:

Try PCA with $k = 1$

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x^{(1)}_{approx}, \ldots, x^{(m)}_{approx}$

Check if

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01?$$

→ [U,S,V] = svd(Sigma)

$S = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & S_{33} & \\ & & & \ddots \\ & & & & S_{nn} \end{bmatrix}$

For given k

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

→ 이렇게 하면 k만 바꿔가면서 해보면 된다. svd는 한번만 소환하면 되기 때문에 편하다.

Choosing k (number of principal components)

→ [U,S,V] = svd(Sigma)

Pick smallest value of k for which

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

(99% of variance retained)

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

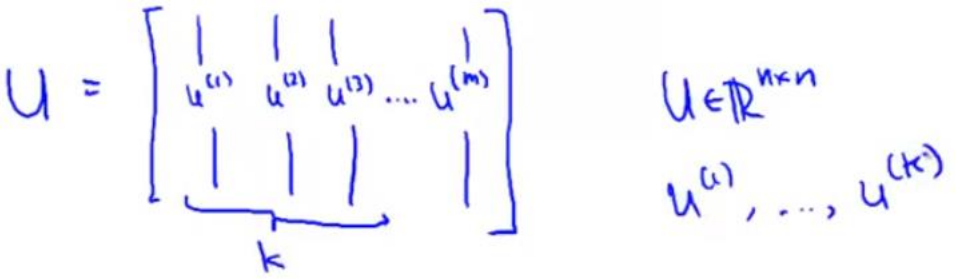
$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T$$

Compute "eigenvectors" of matrix Σ:

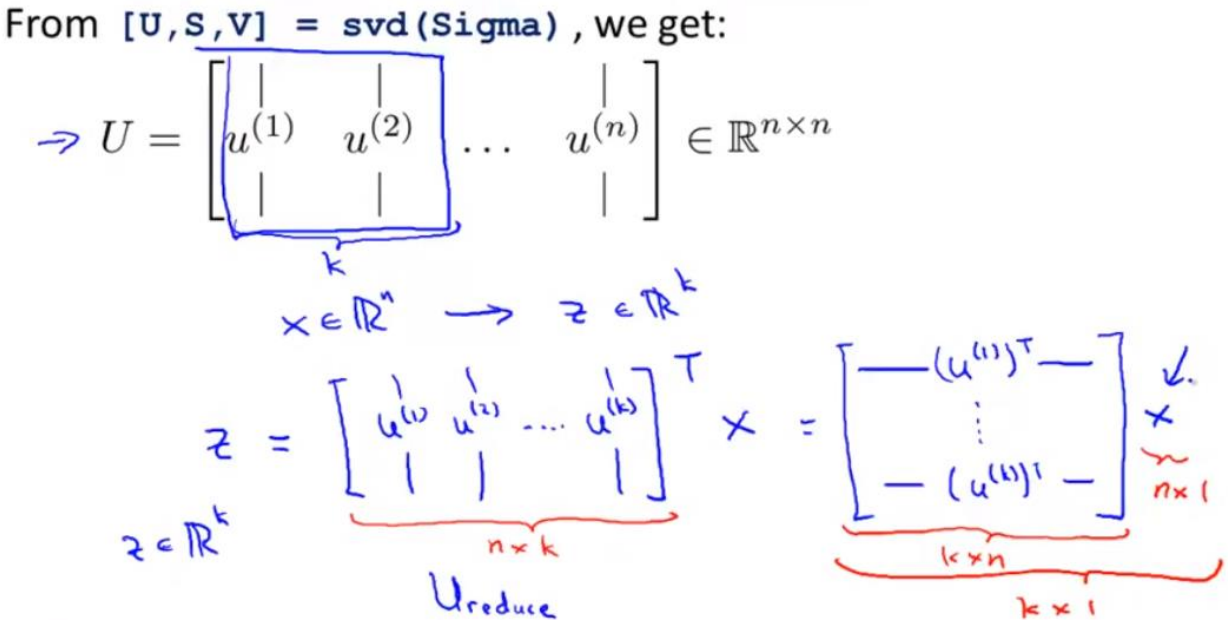
```
[U,S,V] = svd(Sigma);
```

➔ (혁동님) covariance matrix이지 않을까! x

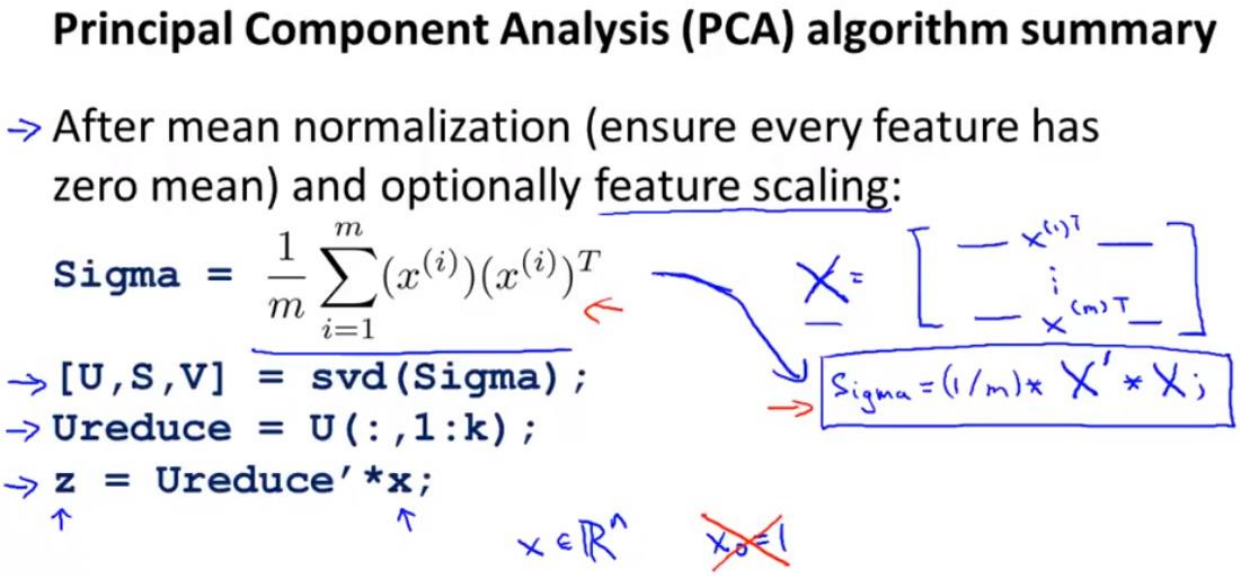
→ sigma라는 변수에 시그마 행렬을 assgin한다. Singular value decomposition 혹은 eig(Sigma)와 거의 같다. svd(Sigma)가 더 stable하니까 이것을 쓰도록. 두 값이 같은 이유는 symmetric positive definite 때문이다. Sigma 행렬은 n x n matrix이다.



→ U matirx는 n x n인데, 그 중에서 k개의 column vector만 선택하면 된다. → U reduce Z 행렬을 새롭게 정의할 수도 있다.



- 정리 :



→ 질문!!! 왜 x(i)랑 x(i)T를 곱한 것이 Sigma가 되는 걸까?

→ 결론적으로 얻는 것은 z vector 들을 모은 Z 행렬이다.

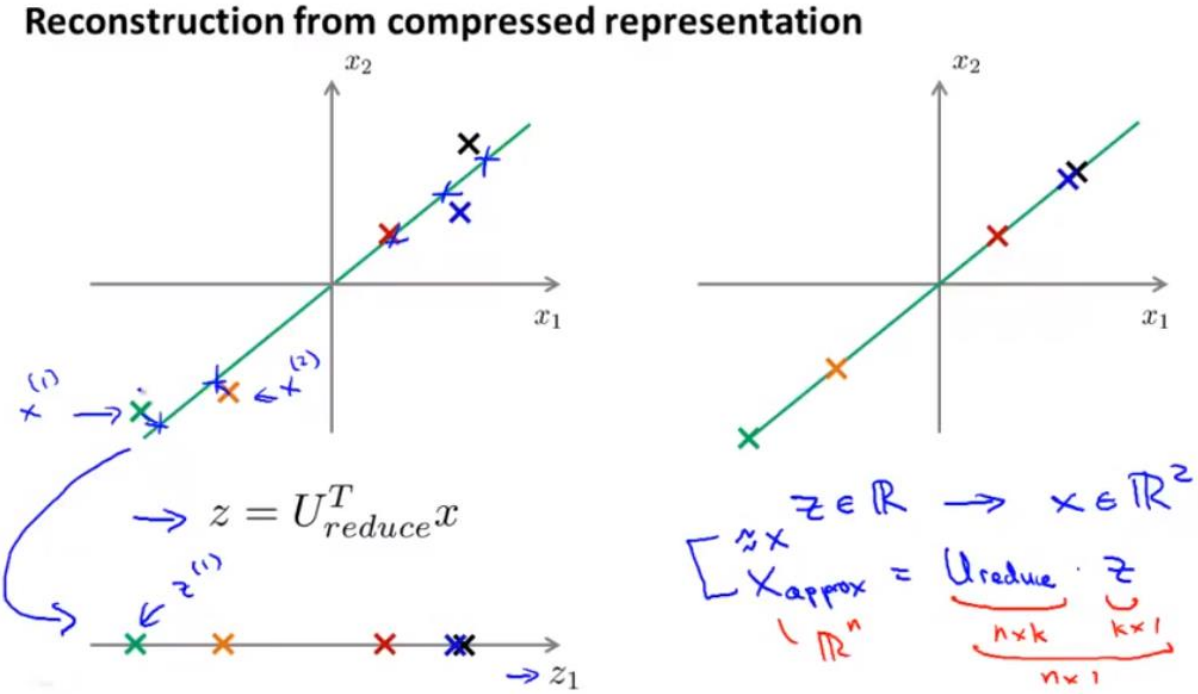
- $z_j = (u^{(j)})^T x$

Correct

2.3.3. PCA Application

(1) Reconstruction from Compressed representation

- 문제 정의 : z가 1차원인데, 다시 x (2차원)으로 되돌아 갈 수 있을까?



- 문제 해결 : X approx. = U reduce * Z

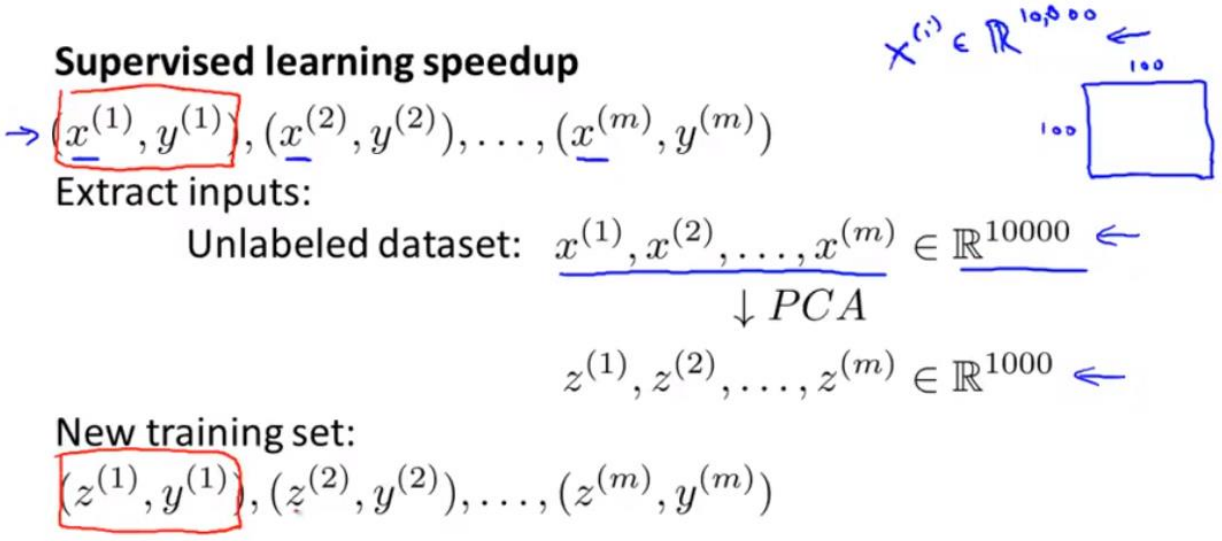
(2) (가장 많이 사용) PCA application-2 : Supervised learning speedup

➔ 원래는 unsupervised에 사용되었는데!! supervised에 적용이 된다!!! 대단!!!!

: extract unlabeled dataset (x vectors)

→ PCA를 통해서 reduced unlabeled dataset (z vectors)

➔ new training set로!!! 만든다. (labeled)



→ dimension이 굉장히 줄어든 z data set을 얻을 수 있다.

- 핵심 : x → z (by U reduce)로 mapping이 핵심이다. (PCA를 통한) 단, training set을 통해서만! 새로운 example은 U reduce를 통해서 z로 변환한 다음에 넣으면 된다.

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets.

x → z

→ 예를 들면 classification을 하면 PCA로 data set의 dimension을 줄인다고 하더라도 크게 정확도가 떨어지지는 않는다.

- 목적에 따른 K의 설정

Application of PCA

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm ←

Choose k by % of variance retain

- Visualization

k=2 or k=3

2.3.4. 주의 사항 (Bad application)

(1) Don't use PCA to reduce overfitting

- 만약에 PCA를 하면 y 에 있는 valuable information이 없어질 가능성이 있다. 물론 99% 이상의 variance가 obtained 되면 큰 문제는 없겠지만 99%를 넘으면 regularization을 쓰는게 overfitting문제를 막는데 더욱 부작용이 적다.

Bad use of PCA: To prevent overfitting

→ Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$.

Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

(2) Always try to do without PCA first.

- PCA없이 해보고, 만약에 PCA가 필요하면(data compression / speed up, visualization) 하는 게 낫다.

PCA is sometimes used where it shouldn't be

Design of ML system:

- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- ~~Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$~~
- Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$
- Test on test set: Map $x_{test}^{(i)}$ to $z_{test}^{(i)}$. Run $h_\theta(z)$ on $\{(z_{test}^{(1)}, y_{test}^{(1)}), \ldots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

→ How about doing the whole thing without using PCA?

→ Before implementing PCA, first try running whatever you want to do with the original/raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$.

III. Special topics of Machine learning

3. Anomaly Detection

3.1. What is Anomaly detection?

- 특징 : unsupervised, supervised의 두가지 특징을 모두 가지고 있다.
- Anomaly Detection이 무엇인가?

(예시1) Fraud Detection

Anomaly detection example

→ Fraud detection:

→ $x^{(i)}$ = features of user i's activities

→ Model $p(x)$ from data.

Identify unusual users by checking which have $p(x) < \varepsilon$

x_1
x_2
x_3
x_4

→ flag users that are behaving unusually.

(예시2) Manufacturing

1) 문제 : 새로운 엔진을 만들었을 때, 그것을 소비자에게 보내도 될 만한 것인가?

Anomaly detection example

Aircraft engine features:

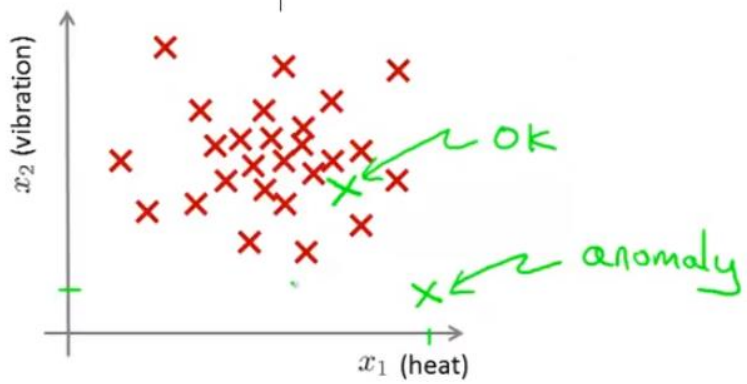
→ x_1 = heat generated

→ x_2 = vibration intensity

...

Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

New engine: x_{test}



2) 해결 : Model P(x)를 새로 만든다! P(x) < ε 이면 anomaly라고 판단하자!

(예시3) Monitoring computers in a data center

→ Monitoring computers in a data center.

→ $x^{(i)}$ = features of machine i

x_1 = memory use, x_2 = number of disk accesses/sec,

x_3 = CPU load, x_4 = CPU load/network traffic.

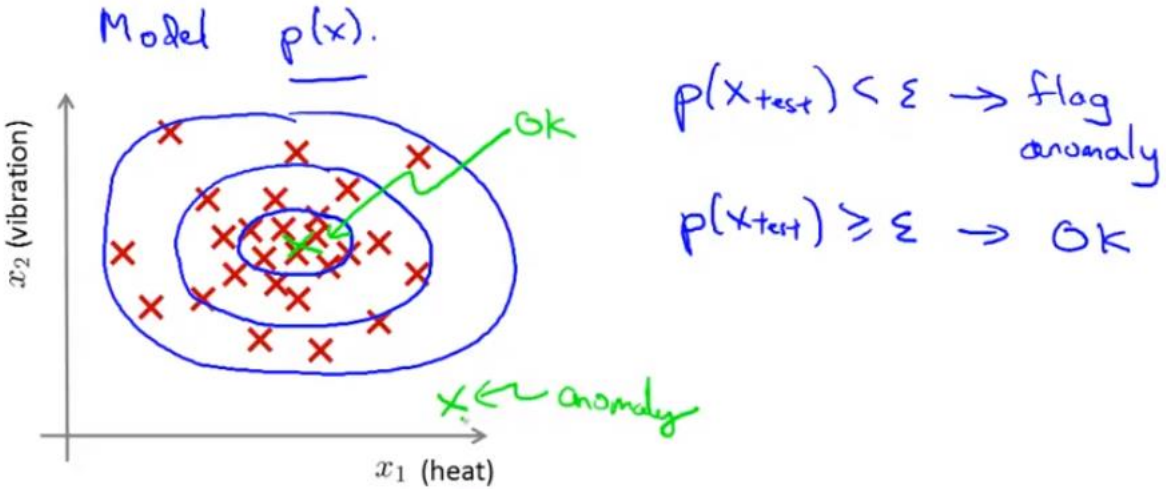
...

$p(x) < \varepsilon$

Density estimation

⇒ Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

⇒ Is x_{test} anomalous?



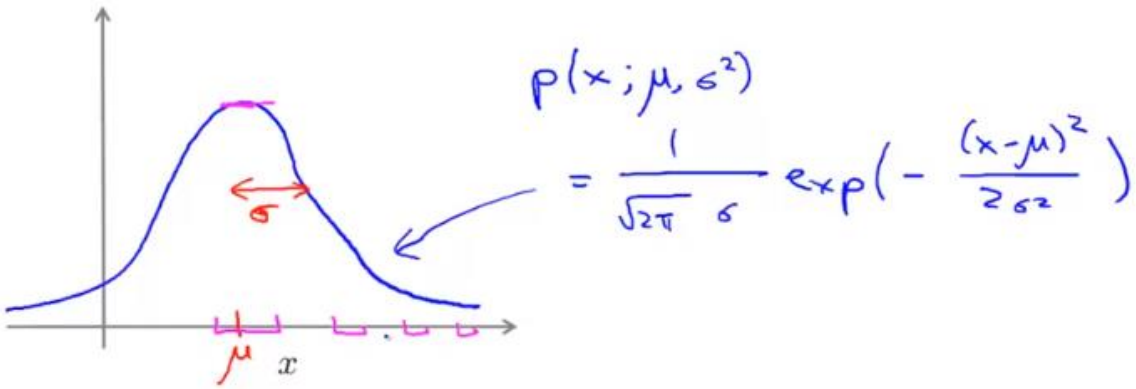
* Gaussian distribution

Gaussian (Normal) distribution

Say $x \in \mathbb{R}$. If x is a distributed Gaussian with mean μ, variance σ^2.

$x \sim \mathcal{N}(\mu, \sigma^2)$

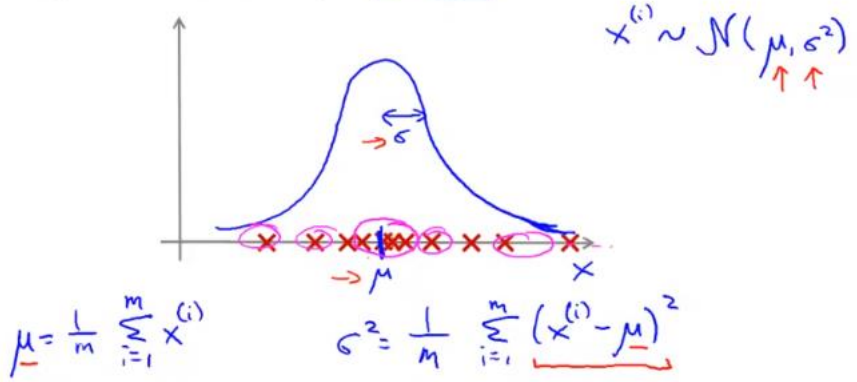
↑ "distributed as"



- Parameter estimation을 해보자 : dataset이 주어지면 어떻게 될 것인가? 이것이 정규분포에서 추출된 것이라고 가정해보고, 정규분포의 평균과 분산을 계산해보면 된다.

Parameter estimation

⇒ Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $x^{(i)} \in \mathbb{R}$



→ maximum likelihood estimation에 해당한다.

- Density estimation : 확률 분포를 정규분포를 예측해보자.

Training set : {x(1), ... , x(m)}

Each example is x ∈ R^n

independent assumption

하면 곱해서 확률을 계산할 수 있다.

P(x) = P(x1; μ1, σ^2) x P2 x P3 x P4

Density estimation

→ Training set: $\{x^{(1)}, \ldots, x^{(m)}\}$

Each example is $x \in \mathbb{R}^n$

$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$

$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$

$x_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$

$$p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2)\, p(x_3; \mu_3, \sigma_3^2) \cdots p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

$\sum_{i=1}^{n} i = 1+2+3+\cdots+n$

$\prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \cdots \times n$

- 계산 방식 :

(1) 일단 주어진 test set으로 평균과 표준편차를 각 독립변수별로 계산한다.

(2) 새로운 것이 anomaly인지 계산한다. P = P1 * P2 < 0.01로 판단하면 된다.

Anomaly detection algorithm

→ 1. Choose features x_i that you think might be indicative of anomalous examples. $\{x^{(1)}, \ldots, x^{(m)}\}$

→ 2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

→ $$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

→ $$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

$p(x_j; \mu_j, \sigma_j^2)$

$\mu_1, \mu_2, \ldots, \mu_n$

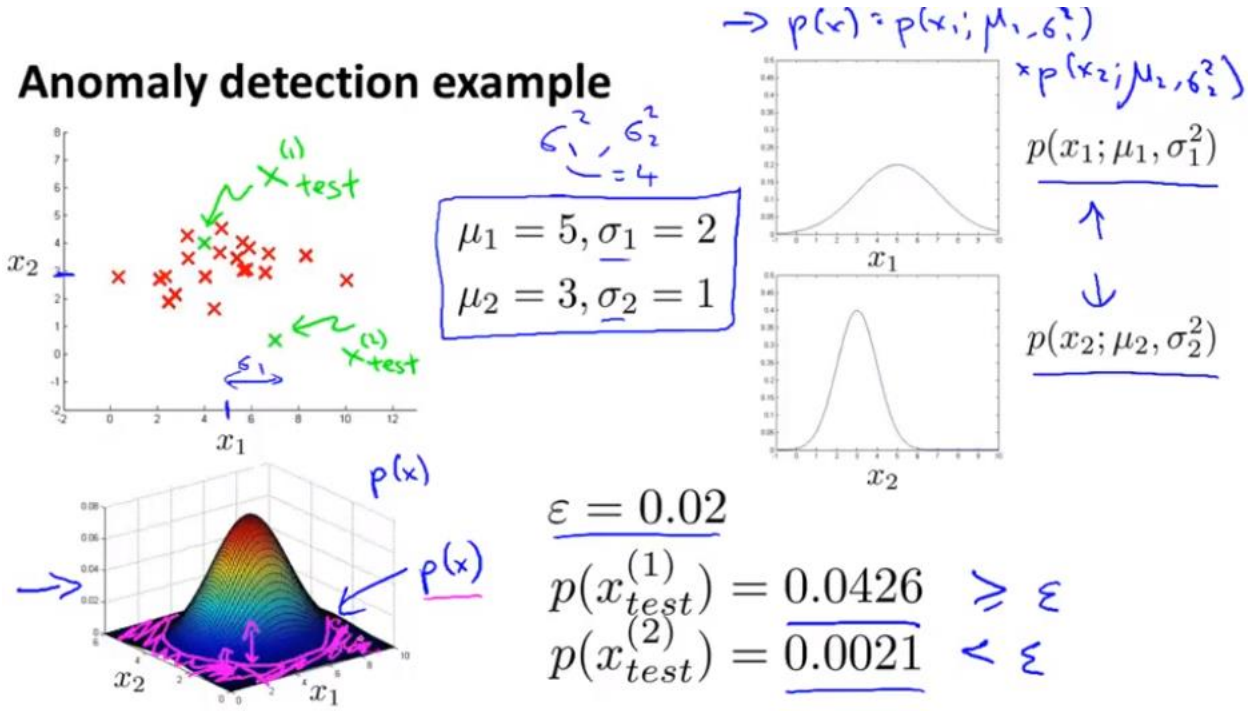
$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

→ 3. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

- 예시 : 성질 변수 x가 2개인 경우의 예를 보자.



→ 이 핑크색으로 표시한 영역에 example이 있게 되면 anomaly가 된다.

- Developing anomaly detection system을 하려면 evaluate를 해서 더 나아지고 있는지 판단할 수 있어야 한다.

The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

→ Assume we have some labeled data, of anomalous and non-anomalous examples. ($y = 0$ if normal, $y = 1$ if anomalous).

→ Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (assume normal examples/not anomalous)

→ Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

→ Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

→ 특이사항 : training set에 anomalous한 값들이 조금은 있어도 된다. 그대로 대부분은 normal한 예여야 한다.

- Training set / cross validation set / test set의 구분

Aircraft engines motivating example

10000 good (normal) engines

20 flawed engines (anomalous) 2-50 y=1

Training set: 6000 good engines ($y = 0$) $p(x) = p(x_1; \mu_1, \sigma_1^2) \cdots p(x_n; \mu_n, \sigma_n^2)$

CV: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Test: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Alternative:

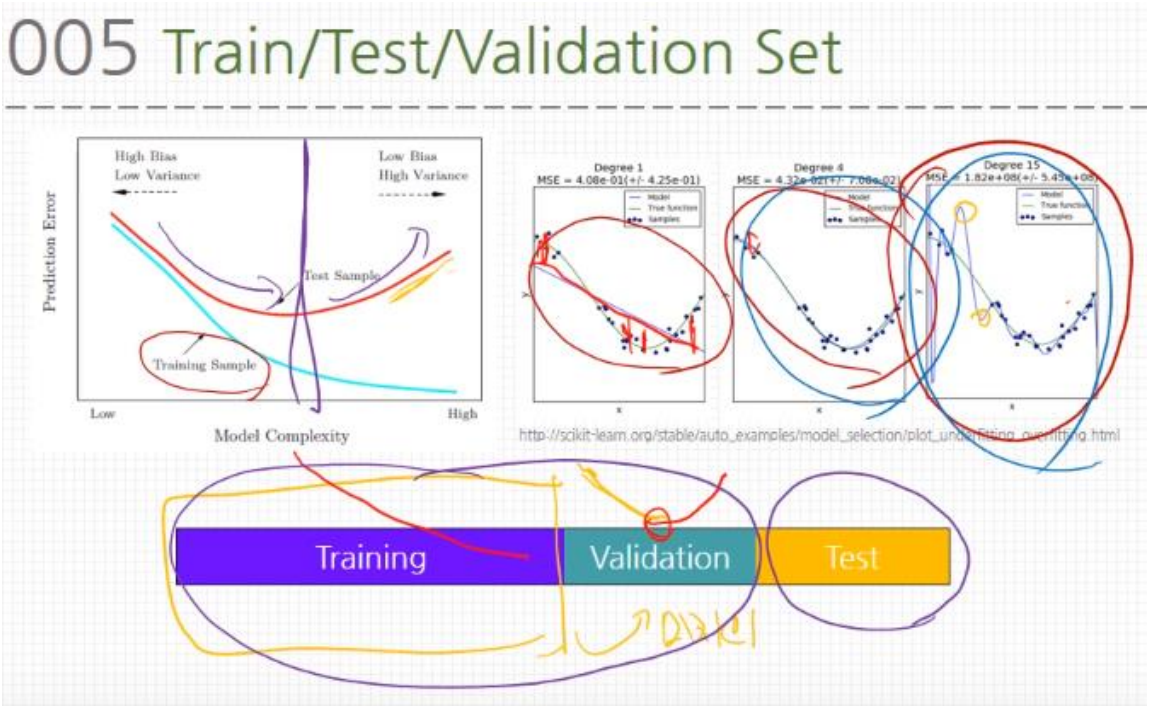
Training set: 6000 good engines

CV: 4000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Test: 4000 good engines ($y = 0$), 10 anomalous ($y = 1$)

→ CV랑 test set을 같은 set으로 하는 것은 문제가 있을 수 있다.

- Train, Test, Validation set을 사용하는 방법 (https://www.youtube.com/watch?v=GtLe9Z2No28)
  - 목적 : Overfitting을 피할 수 있다.
  - 내용 : 학습은 training set으로만 하고, training set이 점점 늘어날수록 fitting error는 줄어들 것이다. 그러나 validation error가 줄다가 저점을 찍고 올라오는 지점이 있을 것이다. 거기서 stop을 해서 training set과 validation set의 황금비율을 찾으면 된다.
  - 선정 : test set이 validation set을 잘 대표할 수 있어야 한다.



- anomaly detection 과정 예시 :

(1) training set으로 p(x)를 만들고 (by 평균, 표준편차 계산) → 그러면 각 x i 별로 y i가 나온다.

Algorithm evaluation

→ Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$

→ On a cross validation/test example x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

(2) cross validation, test set으로 y를 prediction한다. 얼마나 맞는지를 위의 training set으로 만든 모델과 비교한다.

Possible evaluation metrics:

- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1-score

→ 엡실론을 정해본다. F1 score가 뭐지?

- 주의 : skewed data를 쓰기 때문에 classification은 좋은 검증방법이 아닐 수 있다.

Suppose you have fit a model p(x). When evaluating on the cross validation set or test set, your algorithm predicts:

$$y = \begin{cases} 1 & \text{if } p(x) \leq \epsilon \\ 0 & \text{if } p(x) > \epsilon \end{cases}$$

Is classification accuracy a good way to measure the algorithm's performance?

- Yes, because we have labels in the cross validation / test sets.
- No, because we do not have labels in the cross validation / test sets.
- No, because of skewed classes (so an algorithm that always predicts y = 0 will have high accuracy).

3.2. 언제 Anomaly detection을 해야 할까? vs Supervised learning(Logistic regression, neural network)

- 대상 : y = 0, y = 1로 labeled data를 얻었는데 supervised learning 이 Anomaly detection보다 낫지 않을까?
- 비교표 (언제 anomaly detection을 쓰는가)
  - sample에서 negative example(y=0)이 많다. → Anomaly detection
  - anomaly의 종류가 예측 불허 할 때 → Anomaly detection

  cf) spam의 경우에는 anomaly의 종류가 많긴 하지만, 그것에 대한 data set이 이미 다 있기 때문에 supervised를 하는 것이 낫다.

Anomaly detection	vs.	Supervised learning
Very small number of positive examples ($y = 1$). (0-20 is common).		Large number of positive and negative examples.
Large number of negative ($y = 0$) examples. $p(x)$		
Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.		Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.
		Spam

- 예시 :

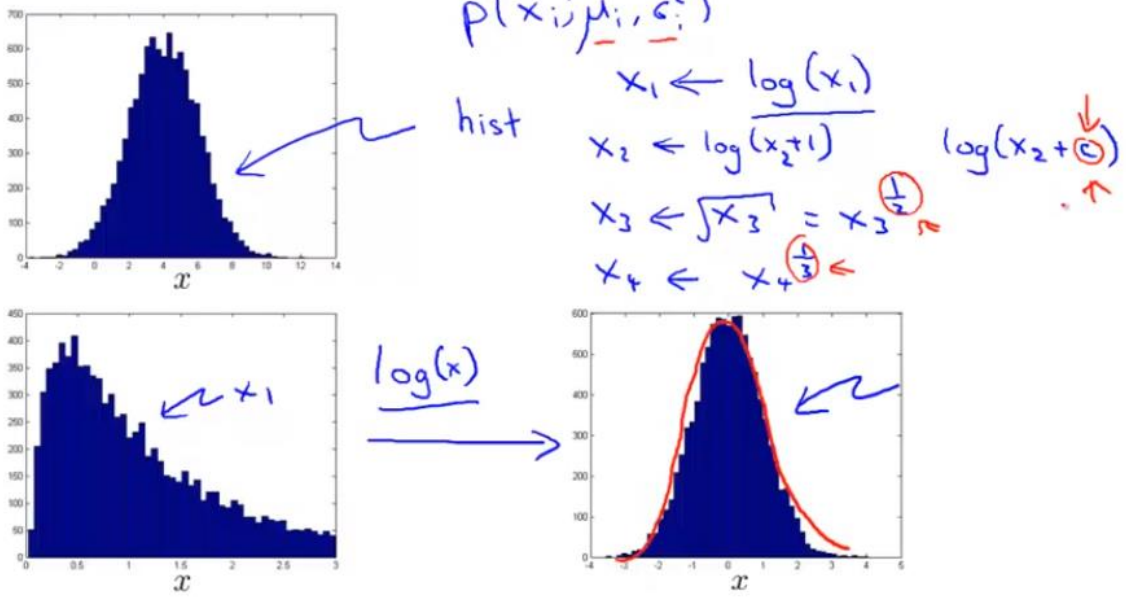
Anomaly detection	vs.	Supervised learning
• Fraud detection $y=1$		• Email spam classification
• Manufacturing (e.g. aircraft engines)		• Weather prediction (sunny/rainy/etc).
• Monitoring machines in a data center		• Cancer classification

3.3. 어떤 feature를 사용해야 할까?

- non-gaussian feature도 사용할 수 있을까? (정규분포가 아니라면??)

(1) 가공을 해서 변수 x를 더 정규분포로 만들 수 있다.

Non-gaussian features



>> hist(x.^0.05, 50)

>> xNew = x.^0.05;

>> hist(log(x), 50);

>> xNew = log(x);

(2) How do you come up with new features of Anomaly detection algorithm

→ Error analysis procedure

(2-1) make new features

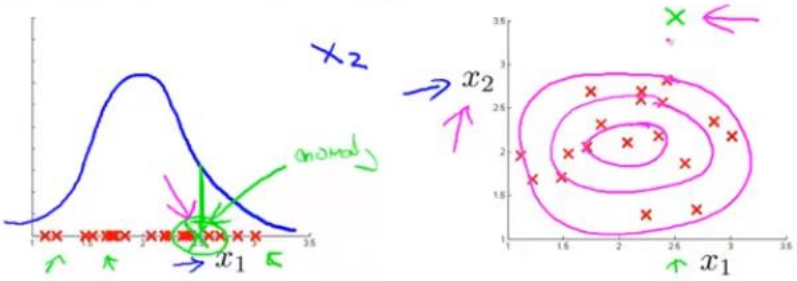
- 문제 상황 : x1의 경우에 anomalous example의 p(x)가 크면, x1 대신에 x2를 찾아본다. 그래서 x1, x2를 동시에 봐서 anomalous example의 p(x)가 작게 끔 한다.

→ **Error analysis for anomaly detection**

Want $p(x)$ large for normal examples x.
$p(x)$ small for anomalous examples x.

Most common problem:

$p(x)$ is comparable (say, both large) for normal and anomalous examples



(2-2) 예시 : 변수 간의 나눗셈을 하면 큰 anomaly를 잡아낼 수 있다.

→ **Monitoring computers in a data center**

→ Choose features that might take on unusually large or small values in the event of an anomaly.

→ x_1 = memory use of computer

→ x_2 = number of disk accesses/sec

→ x_3 = CPU load ←

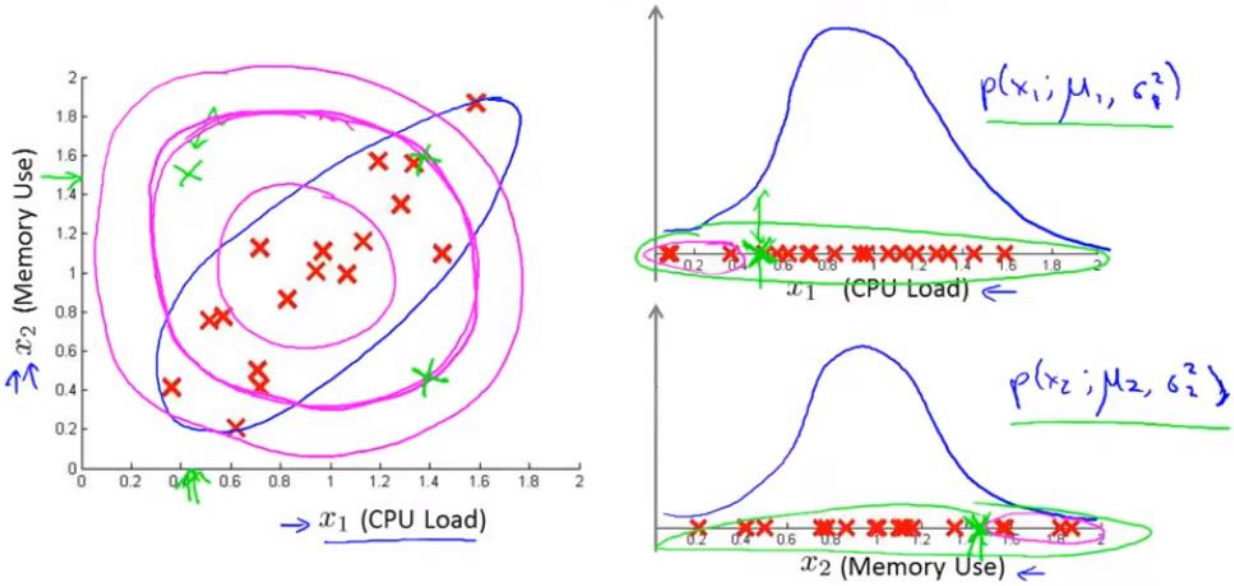
→ x_4 = network traffic ←

$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$

$x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$

3.4. (심화학습) Multivariate Gaussian distribution

Motivating example: Monitoring machines in a data center



→ P(x)를 x1, x2별로 하면 높다. 하지만 x1, x2를 동시에 생각하면 이상값이다.

(혁동님) 결합을 해야 anomaly를 detect할 수 있다!!

블록 비유 : 1, 4, 6, 4, 1으로 대각선으로 쌓여있을 때, 위에서 보면 detection이 가능하다. anomaly를!!

- Multivariate gaussian distribution

Multivariate Gaussian (Normal) distribution

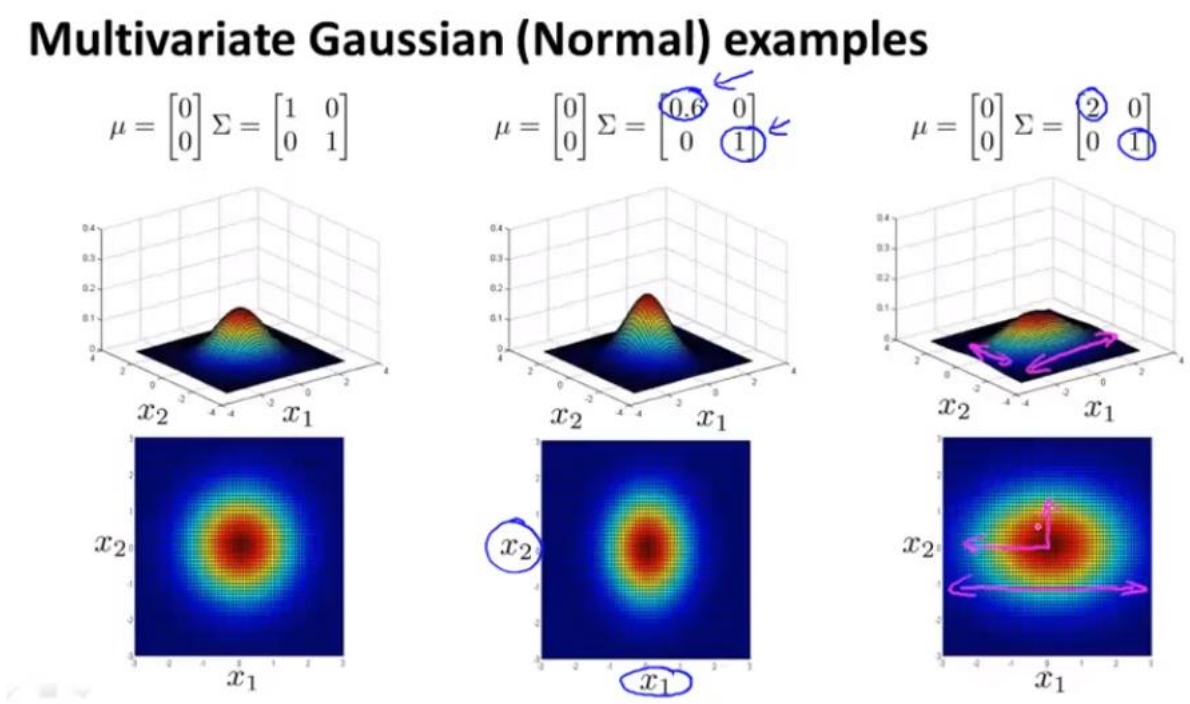
→ $x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2), \ldots,$ etc. separately.
Model $p(x)$ all in one go.
Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

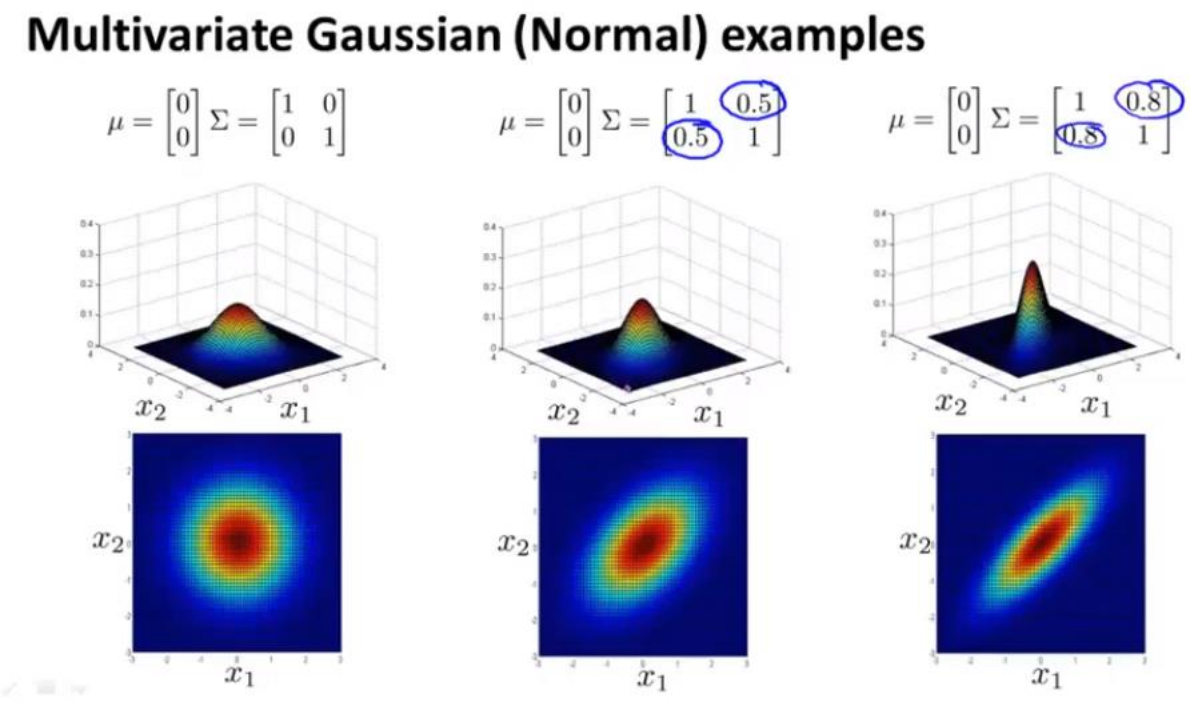
$|\Sigma|$ = determinat of Σ | det (Sigma)

→ 엄청 이상하네 ㅋㅋㅋ 식이.

- variance를 바꾸면 어떻게 편할까?



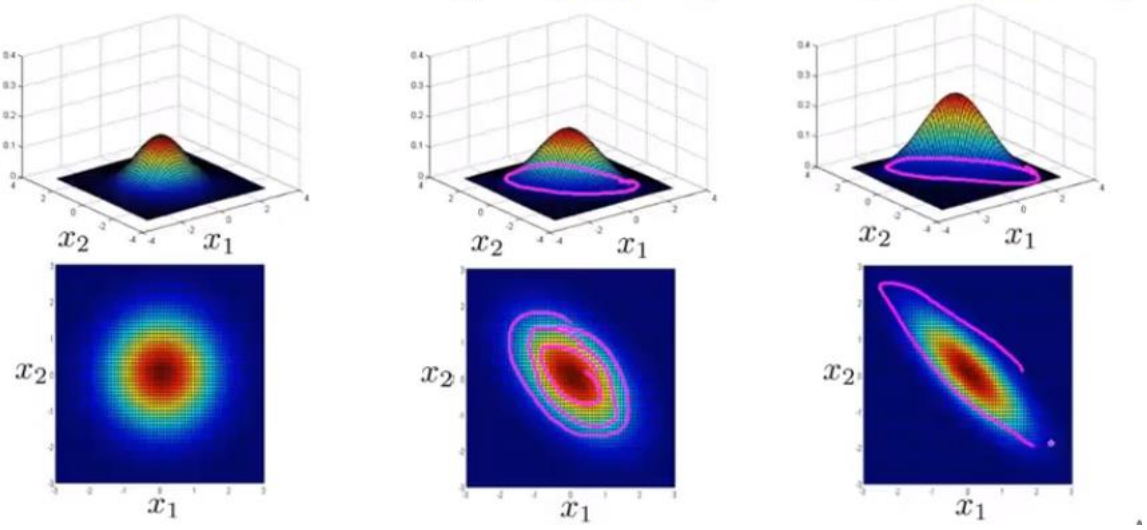
- Correlation이 달라지는 경우. : diagonal side에 있는 걸로!



- negative correlation

Multivariate Gaussian (Normal) examples

$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$



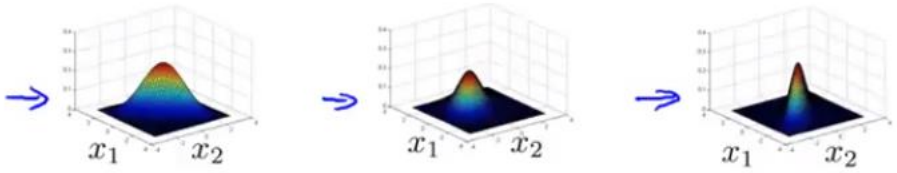
- 실제로 anomaly detection을 하는 방법

(1) 먼저, 평균과 표준편차 행렬(covariance matrix)을 구해서.정규분포를 가정하여 P(x)를 fitting한다.

Multivariate Gaussian (Normal) distribution

Parameters μ, Σ $\mu \in \mathbb{R}^n$ $\Sigma \in \mathbb{R}^{n \times n}$

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$



Parameter fitting:

Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $x \in \mathbb{R}^n$

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

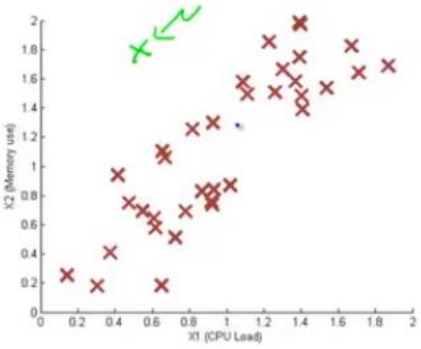
(2) 새로운 sample x의 확률을 구해서 anomaly인지 아닌지 판단한다.

Anomaly detection with the multivariate Gaussian

1. Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$



2. Given a new example x, compute

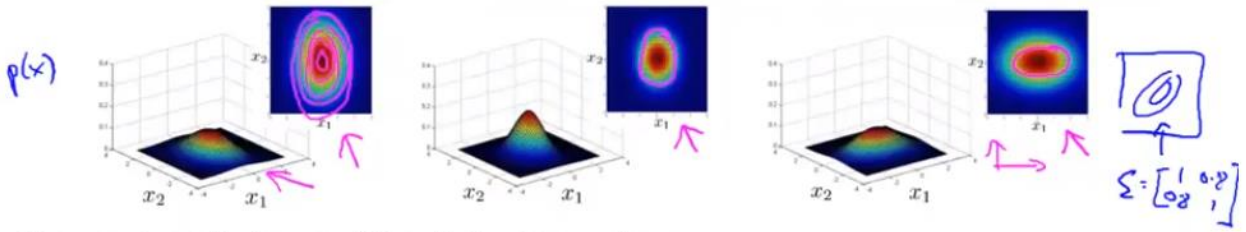
$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Flag an anomaly if $p(x) < \varepsilon$

- multivariate Gaussian model과 original model과의 차이점 : original model은 등고선이 대각 방향으로 향해있을 수가 없다.

Relationship to original model

Original model: $p(x) = p(x_1;\mu_1,\sigma_1^2) \times p(x_2;\mu_2,\sigma_2^2) \times \cdots \times p(x_n;\mu_n,\sigma_n^2)$



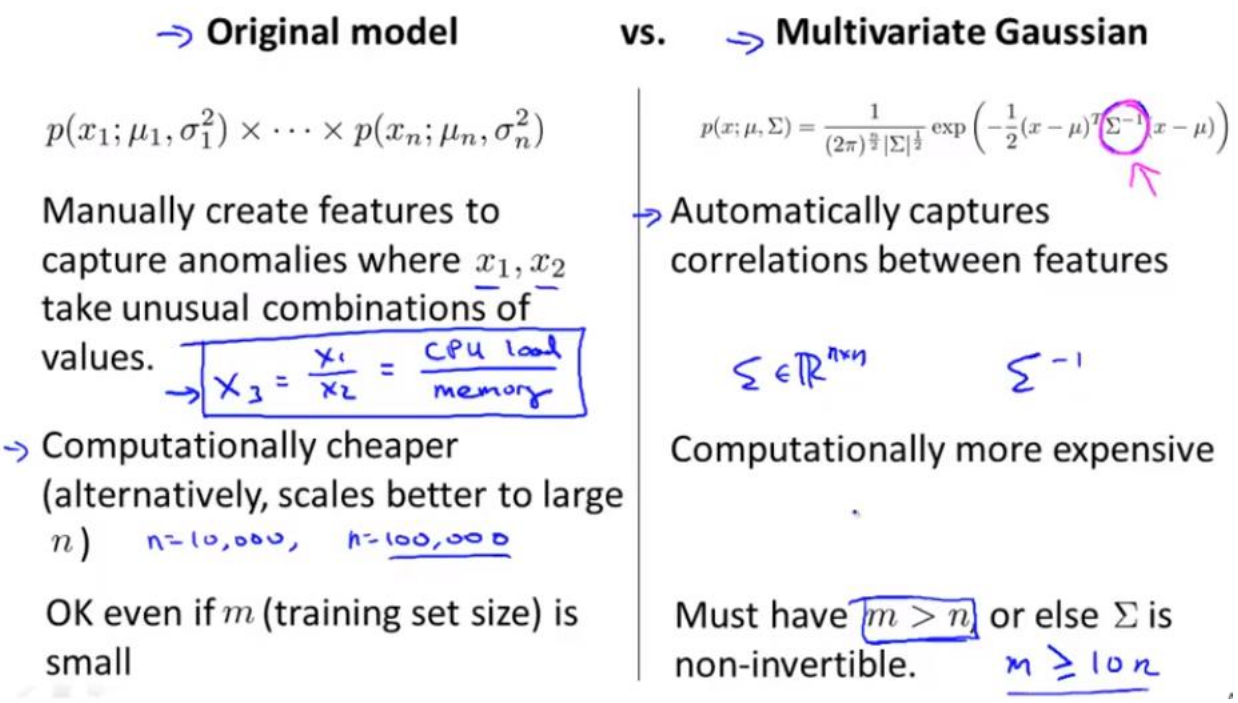
Corresponds to multivariate Gaussian

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

where $\Sigma = \begin{bmatrix} \sigma_1^2 & & & 0 \\ & \sigma_2^2 & & \\ & & \ddots & \\ 0 & & & \sigma_n^2 \end{bmatrix}$

Andrew Ng

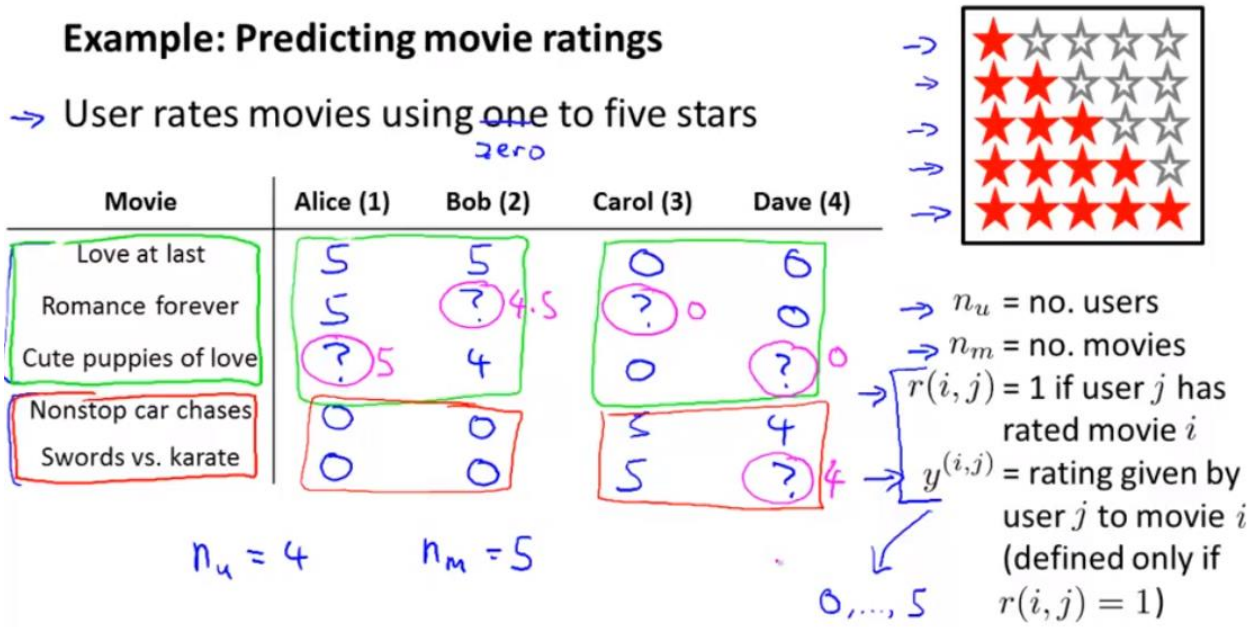
- 언제 original model, 언제 multivariate Gaussian model을 사용해야 할까?
  - original model은 anomaly를 잡기 위해서 새로운 변수 x3를 만들어야 한다. 보통 더 많이 쓴다.
  - multivariate Gaussian model은 역행렬을 구한다는 것이 좀 부담스럽다. 그런데, 변수 개수보다 데이터 양이 훨씬 더 많으면 original 방법보다 편하다. original 방법은 새로운 변수를 더 많이 만들어야 하기 때문이다. 변수가 redundant하면 알고 있는 변수가 줄어드는 효과가 있기 때문에 original을 사용하는 것이 더 좋을 수 있다.



- 변수 2개만 써도 n (n+1) / 2 개가 되니까 너무 많아진다.

4. Recommender system

- 형태 : 영화 추천 시스템



4.1. feature를 먼저 알고 user들의 parameter vector를 구하는 방법

- feature → content based recommendations : 영화의 속성을 넣을 수 있다. 영화의 속성이 평가자의 평가 점수에 미치는 영향을 learning 해보자.
  → 사용자 각각 theta를 정의할 수 있다.

Content-based recommender systems

$n_u = 4$, $n_m = 5$

$x_0 = 1$

$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
Love at last 1	5	5	0	0	0.9	0
Romance forever 2	5	?	?	0	1.0	0.01
Cute puppies of love 3	? 4.95	4	0	?	0.99	0
Nonstop car chases 4	0	0	5	4	0.1	1.0
Swords vs. karate 5	0	0	5	?	0	0.9

$n = 2$

→ For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

$\theta^{(j)} \in \mathbb{R}^{n+1}$

$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$ ↔ $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ $\quad (\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$

- linear regression과 유사하다!

Problem formulation

→ $r(i,j) = 1$ if user j has rated movie i (0 otherwise)

→ $y^{(i,j)}$ = rating by user j on movie i (if defined)

→ $\theta^{(j)}$ = parameter vector for user j

→ $x^{(i)}$ = feature vector for movie i

→ For user j, movie i, predicted rating: $(\theta^{(j)})^T (x^{(i)})$

→ $m^{(j)}$ = no. of movies rated by user j

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

→ regularization term도 있다.

- 방법 정리 :

Optimization algorithm:

$$\min_{\theta^{(1)},\ldots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

$J(\theta^{(1)},\ldots,\theta^{(n_u)})$

Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda\theta_k^{(j)}\right) \quad (\text{for } k \neq 0)$$

→ 1/m 은 optimization식에서 없앴기 때문에 없는거다.

4.2. Collaborative filtering → 어떤 feature를 쓸 지 스스로 배운다.

- 목적 : feature가 주어지지 않으면?? User의 parameter vector를 안 후에 feature 값을 찾아낸다.
- 예시 :

(1) 초기 조건

Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

(2) user들이 취향을 말해줬다.

$$\theta^{(1)} = \begin{bmatrix}0\\5\\0\end{bmatrix}, \theta^{(2)} = \begin{bmatrix}0\\5\\0\end{bmatrix}, \theta^{(3)} = \begin{bmatrix}0\\0\\5\end{bmatrix}, \theta^{(4)} = \begin{bmatrix}0\\0\\5\end{bmatrix}$$

(3) 이를 통해서 x1, x2를 알아낼 수 있다. 그들이 한 (평가 점수) = theta^T * feature

- 알고리즘 :
  - 아까는 θ으로 minimize를 했는데, 이제는 x i를 통해 식을 minimize한다.

Optimization algorithm

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

$$\min_{x^{(1)},\ldots,x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

- $x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$

→ θ가 주어졌는지, x가 주어졌는지에 따라서 방법이 달라진다.

- 두가지를 번갈아 가면서 하면 된다.

Collaborative filtering

Given $x^{(1)}, \ldots, x^{(n_m)}$ (and movie ratings),
can estimate $\theta^{(1)}, \ldots, \theta^{(n_u)}$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$,
can estimate $x^{(1)}, \ldots, x^{(n_m)}$

Guess $\Theta \rightarrow X \rightarrow \Theta \rightarrow X \rightarrow \Theta \rightarrow X \rightarrow \cdots$

- Collaborative filtering 알고리즘
  - theta랑 x를 번갈아하지 않아도 된다.
    → cost function을 합치면 된다.

Collaborative filtering optimization objective

→ Given $x^{(1)}, \dots, x^{(n_m)}$, estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)},\dots,\theta^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

→ Given $\theta^{(1)}, \dots, \theta^{(n_u)}$, estimate $x^{(1)}, \dots, x^{(n_m)}$:

$$\min_{x^{(1)},\dots,x^{(n_m)}} \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$$

Minimizing $x^{(1)}, \dots, x^{(n_m)}$ and $\theta^{(1)}, \dots, \theta^{(n_u)}$ simultaneously:

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

$$\min_{x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$$

→ 하늘색, 빨간색 부분이 각각 θ, x를 기준으로 할 때, 다른 것이 상수가 되므로 합쳐도 무방하다.

→ 주의사항 : x, θ에는 0항이 포함되지 않기 때문에 항개수가 n+1이 아니라, n이다.

- 정리 :

Collaborative filtering algorithm

1. Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values.
2. Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$ using gradient descent (or an advanced optimization algorithm). E.g. for every $j = 1, \dots, n_u, i = 1, \dots, n_m$:

$$x_k^{(i)} := x_k^{(i)} - \alpha\left(\sum_{j:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})\theta_k^{(j)} + \lambda x_k^{(i)}\right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha\left(\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda\theta_k^{(j)}\right)$$

3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

→ 1. 처음에 small random value로 initialize하는 이유 : this serves as a symmetry breaking and ensures the algorithm learns feature that are different from each other.

- Vectorization : Low rank matrix factorization :
  - predicted rating을 하는 방법 : matrix를 만들면 된다.

Collaborative filtering

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

Predicted ratings:

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

-

Collaborative filtering

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

Predicted ratings:

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix} \qquad \Theta = \begin{bmatrix} -(\theta^{(1)})^T- \\ -(\theta^{(2)})^T- \\ \vdots \\ -(\theta^{(n_u)})^T- \end{bmatrix}$$

Low rank matrix factorization

Let $X = \begin{bmatrix} - & (x^{(1)})^T & - \\ & \vdots & \\ - & (x^{(n_m)} & - \end{bmatrix}$, $\Theta = \begin{bmatrix} - & (\theta^{(1)})^T & - \\ & \vdots & \\ - & (\theta^{(n_u)} & - \end{bmatrix}$.

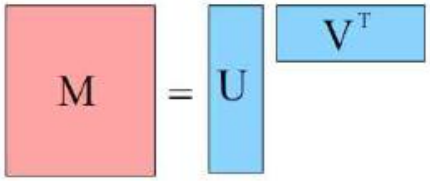
What is another way of writing the following:

$$\begin{bmatrix} (x^{(1)})^T(\theta^{(1)}) & \dots & (x^{(1)})^T(\theta^{(n_u)}) \\ \vdots & \ddots & \vdots \\ (x^{(n_m)})^T(\theta^{(1)}) & \dots & (x^{(n_m)})^T(\theta^{(n_u)}) \end{bmatrix}$$

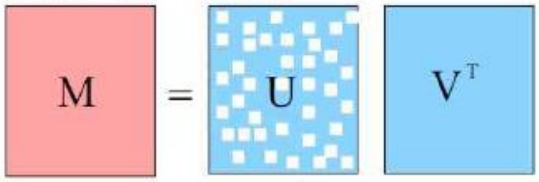
- low rank라는 것은 linearly independent한 vector의 수이다. 그것을 줄이면 연산이 편해진다.
(혁동님 자료)

Two types of sparsity for matrices $\mathbf{M} \in \mathbb{R}^{n \times p}$
II - Through a factorization of $\mathbf{M} = \mathbf{U}\mathbf{V}^\top$

- Matrix $\mathbf{M} = \mathbf{U}\mathbf{V}^\top$, $\mathbf{U} \in \mathbb{R}^{n \times k}$ and $\mathbf{V} \in \mathbb{R}^{p \times k}$
- **Low rank**: m small



- **Sparse decomposition**: U sparse



6)

Item

User	W	X	Y	Z
A		4.5	2.0	
B	4.0		3.5	
C		5.0		2.0
D		3.5	4.0	1.0

Rating Matrix

=

A	1.2	0.8
B	1.4	0.9
C	1.5	1.0
D	1.2	0.8

User Matrix

×

1.5	1.2	1.0	0.8
1.7	0.6	1.1	0.4

Item Matrix

Item

User	W	X	Y	Z
A		4.5	2.0	
B	4.0		3.5	
C		5.0		2.0
D		3.5	4.0	1.0

Rating Matrix

=

A	1.2	0.8	0.8
B	1.4	0.9	0.9
C	1.5	1.0	1.0
D	1.2	0.8	0.8

User Matrix

×

1.5	1.2	1.0	0.8
1.7	0.6	1.1	0.4
1.7	0.6	1.1	0.4

Item Matrix

→ 더 적은 양이 연산으로!!

Finding related movies

For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$.

x_1 = romance, x_2 = action, x_3 = comedy, x_4 = ...

- how to find related movies??

How to find movies j related to movie i?

small $\|x^{(i)} - x^{(j)}\|$ → movie j and i are "similar"

5 most similar movies to movie i:

→ Find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.

→ 거리로 알 수 있다.

- mean normalization :

- 문제상황 : 만약에 어떤 사람이 영화평가를 1개도 안 했다고 하면, 다 0으로 rating할 것으로 예측이 되어버린다. 그 이유는 cost function의 첫 항은 정보가 없어서 무용지물이고, regularization term에서 θ를 각각 다 0으로 해야지 cost function이 최소화되기 때문이다.

Users who have not rated any movies

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\min_{\substack{x^{(1)},\dots,x^{(n_m)} \\ \theta^{(1)},\dots,\theta^{(n_u)}}} \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$n=2$ $\theta^{(5)} \in \mathbb{R}^2$ $\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\frac{\lambda}{2}\left[(\theta_1^{(5)})^2 + (\theta_2^{(5)})^2\right]$ ←

$(\theta^{(5)})^T x^{(i)} = 0$

- Mean normalization
  - average를 따로 저장해놓는다.
  - 평가를 한 거에서 평균을 빼버린다. → 평균이 0이 되게 만든다.
  - predict를 하고 나서 평균을 더하면 최종 예측값이 되게 한다.

Mean Normalization:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \quad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

learn $\Theta^{(j)}, x^{(i)}$

For user j, on movie i predict:

$$(\Theta^{(j)})^T (x^{(i)}) + \mu_i$$

User 5 (Eve):

$$\Theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\underbrace{(\Theta^{(5)})^T (x^{(i)})}_{=0} + \mu_i$$

IV. What Machine Learning should I apply?

- 이 단원이 굉장히 중요한 단원이 될 것이다. Debugging과정과 어디서 머신 러닝 개선을 이룰 수 있는지를 알 수 있게 해준다.

- How to improve the learning algorithm

1. get more training examples
   - 문제점 : 2배를 더 모으면 개선될까? 아닐 수 있다. 시간 낭비일 수 있다.
2. feature가 너무 많을 때는 overfitting이 일어날 수 있으므로 feature를 몇 개만 선정한다.
3. feature를 더 늘린다.
4. polynomial feature (x^2, x1*x2)
5. decrease/increase "lambda" regularization parameter

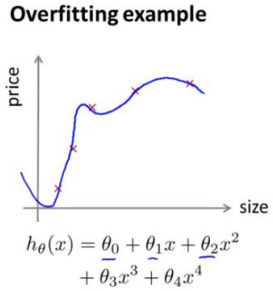
- 위의 option 중 어떤 것을 하는 게 좋을지 정하는 방법이 있다.

➔ **Machine learning diagnostic** : a test that you can run to gain insight what is / isn't working with a learning algorithm.

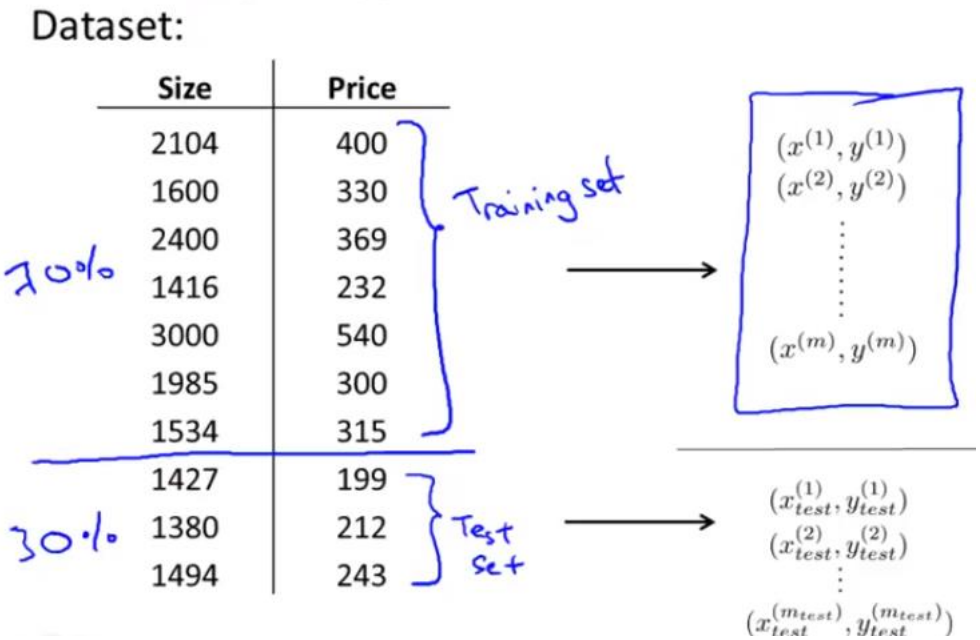
6. Machine Learning diagnostic

6.1. Test set을 배정하여 test error를 보면 overfitting인지 아닌지 볼 수 있다.

- How to evaluate a hypothesis : **training error** & test error를 둘 다 측정한다!!
  - training error가 low한지는 좋은 지표가 아니다. → overfitting의 문제



  - training set(70%) + test set(30%)를 해서 확인할 수 있다.
  - test set error가 high하면 overfitting한 것이다.



(1) linear regression

Training/testing procedure for linear regression

- Learn parameter θ from training data (minimizing training error $J(\theta)$) 70%

- Compute test set error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2$$

(2) logistic regression 도 test set을 사용한다!

- test error 도 좋고, Misclassification error도 계산한다.

- Compute test set error:

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log h_\theta(x_{test}^{(i)})$$

- Misclassification error (0/1 misclassification error):

$$err(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, \; y = 0 \\ & \text{or if } h_\theta(x) < 0.5, \; y = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{error}$$

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}), y_{test}^{(i)})$$

6.2. Validation set을 배정하여 polynomial model 중에 어떤 것이 진정으로 나은지 선택할 수 있다! (polynomial degree를 fit하는데 test set을 이미 써버렸다!)

- model selection problem

Model selection

1. $h_\theta(x) = \theta_0 + \theta_1 x$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$

   $\vdots$

10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$

- degree of polynomial / choosing regularization parameter
- 어떤 model을 골라야 할까?

Model selection

d = degree of polynomial

1. $h_\theta(x) = \theta_0 + \theta_1 x$ → $\Theta^{(1)}$ → $J_{test}(\Theta^{(1)})$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ → $\Theta^{(2)}$ → $J_{test}(\Theta^{(2)})$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$ → $\Theta^{(3)}$ → $J_{test}(\Theta^{(3)})$

⋮

10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$ → $\Theta^{(10)}$ → $J_{test}(\Theta^{(10)})$

Choose $\theta_0 + \ldots \theta_5 x^5$ ←

How well does the model generalize? Report test set error $J_{test}(\theta^{(5)})$.

Problem: $J_{test}(\theta^{(5)})$ is likely to be an optimistic estimate of generalization error. I.e. our extra parameter (d = degree of polynomial) is fit to test set.

→ 이 예에서는 test set을 가지고 degree of polynomial을 결정했기 때문에 d=5일 때가 가장 좋은 예측 모델이라고 하기가 힘들다는 것이다 → 그러면 training set을 가지고 하면 되는 것 아니냐? 아 Theta를 training set으로 유도하는 것이다!!! 그 다음에 test set으로 10개 모델 중에서 어떤 게 가장 나은지를 본 것이다. 근데 d=5가 나은 걸 또 검증해야 하는데.... 이 포인트에서 validation set이 나온다!

- 해결 : 그러면 data set을 1) training set (60%) 2) **(cross) validation set (cv)** (20%) 3) test set (20%)로 세가지로 나누면 되지 않느냐?

Evaluating your hypothesis

Dataset:

	Size	Price	
60%	2104	400	Training set
	1600	330	
	2400	369	
	1416	232	
	3000	540	
	1985	300	
20%	1534	315	Cross validation set (CV)
	1427	199	
20%	1380	212	test set
	1494	243	

$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
⋮
$(x^{(m)}, y^{(m)})$

$(x_{cv}^{(1)}, y_{cv}^{(1)})$
$(x_{cv}^{(2)}, y_{cv}^{(2)})$
⋮
$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

m_{cv} = no. of cv examples $(x_{cv}^{(i)}, y_{cv}^{(i)})$

$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
⋮
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test}

→ training error, cross validation error, test error 세가지의 error가 나오게 된다.

Training error:

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

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

→ 여기서 validation set을 가지고 d를 선택하고, 그 다음에 남겨진 test set을 가지고 d=5를 검증하는데 사용하면 되지 않느냐.

- 정리 :

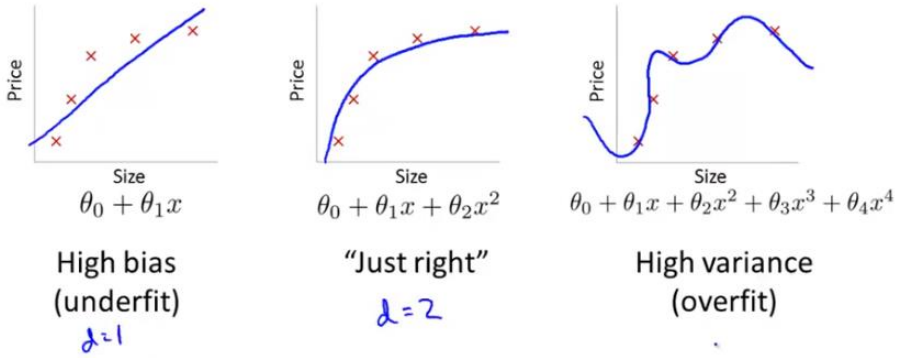
We can now calculate three separate error values for the three different sets using the following method:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);

→ 여기서는 validation set을 가지고 polynomial degree d를 fitting했다.

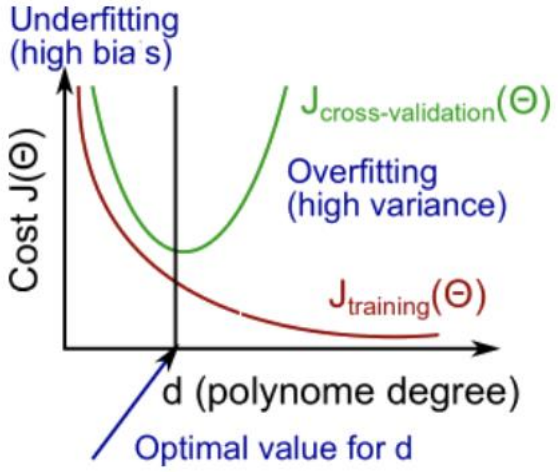
6.3. feature가 많아서 문제냐, 적어서 문제냐! Diagnosing bias vs. variance!!

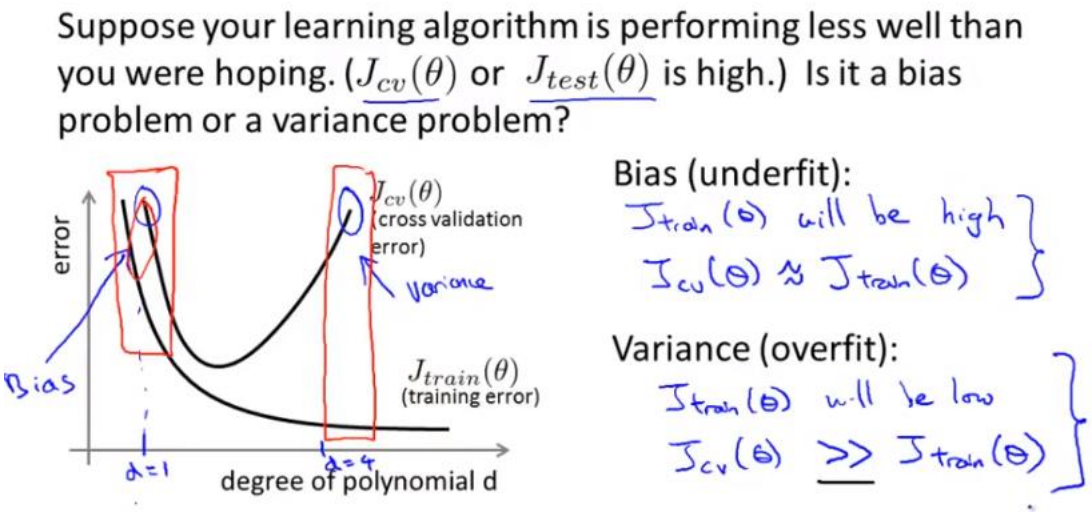
- 문제 : feature가 너무 많은 경우, feature가 부족한 경우를 어떻게 알까?
- high bias problem → underfitting problem (bias는 하나의 일직선으로 생각하자)
  high variance problem → overfitting problem (variance는 구불구불한 선으로 생각하자)



→ d가 오른쪽으로 갈수록 커진다.

- 해결 : training set error가 작으면 이건 overfitting(high variance)이다. cross validation error는 over나 under나 둘 다 높다. 왜냐하면 최적의 degree of polynomial이 있기 때문이다.



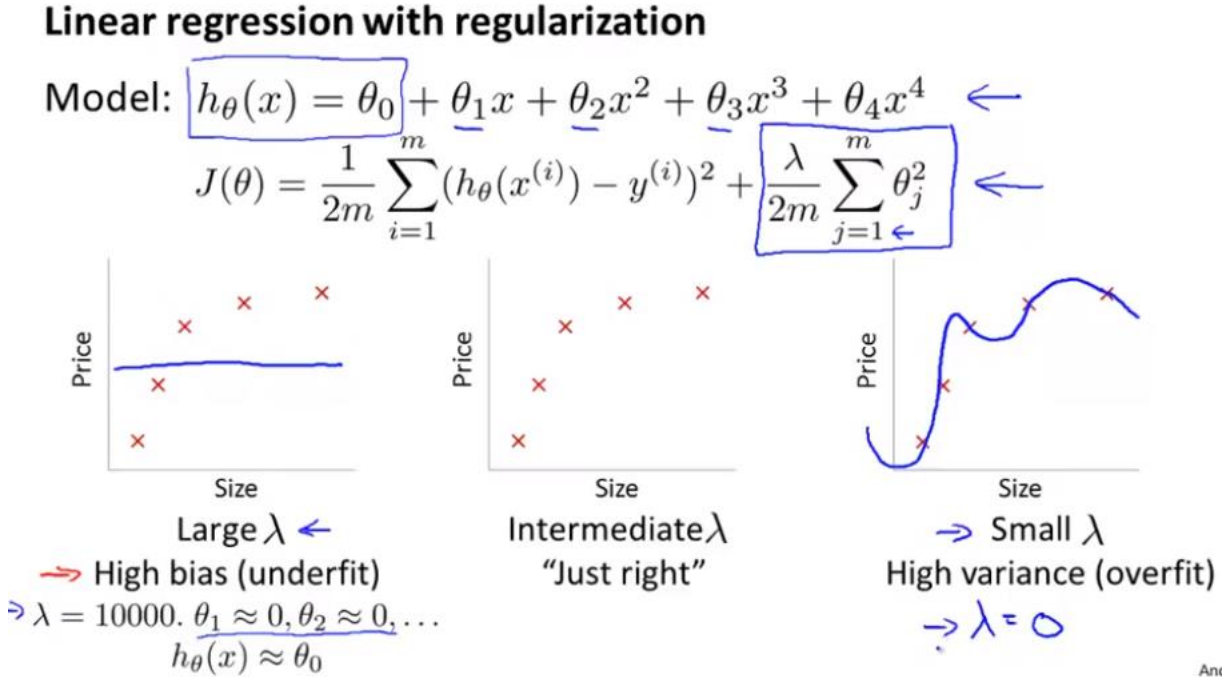


(1) Bias (underfit) : training error가 높고 validation set error가 높아서 둘이 비슷한 경우.

(2) Variance (overfit) : training error가 낮고, validation set error가 높은 경우. 따라서 cross validation error가 training set error보다 훨씬 큰 경우

6.4. Lambda (regularization parameter)을 정할 때는 마찬가지로 cross validation set을 사용한다!!

- regularization이 어떻게 bias이랑 variance에 영향을 미치는지?
  - 이 얘기는 예전에 이미 했던 것이다.
  - Lambda (regularization parameter)가 크면 theta가 penalize되기 때문에 Theta 0만이 남게 된다. 따라서 high bias가 된다.
  - 만약 Lambda 가 작게 되면 Theta가 penalize가 적게 되니까, overfitting이 된다.



- 어떻게 Lambda를 정할 것인가?

Choosing the regularization parameter λ

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

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

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

→ training error , validation error, test error에는 regularization term이 없다. regularization term에서 m이 아니라 n이다.

- Lambda를 정하기 위한 model selection 방법

(1) 일단 cost function J를 최소화하는 theta를 구한다. 이 식에는 물론 lambda가 포함되어 있다.

(2) 다음, cross validation set을 통해서 어떤 Theta vector가 가장 cost function이 낮은지 본다. 여기에는 regularization term이 없다.

(3) test set으로 마지막에 다시 점검한다.

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

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

1. Try $\lambda = 0$
2. Try $\lambda = 0.01$
3. Try $\lambda = 0.02$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$

⋮

12. Try $\lambda = 10$

Pick (say) $\theta^{(5)}$. Test error:

- Training error와 cross validation error를 lambda에 대해 그래프를 그려보자.

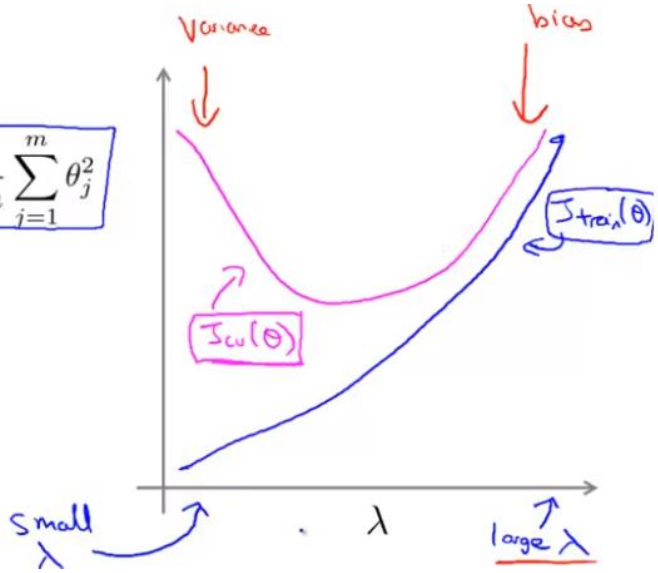
(1) Training error는 lambda가 커지면 bias가 커지게 되기 때문에(underfitting) training error가 커진다.

(2) Validation error는 lamda가 특정값(optimum)일 때를 제외하고는 양쪽으로 커질 것이다.

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

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

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



- Lambda 고르기 과정 정리

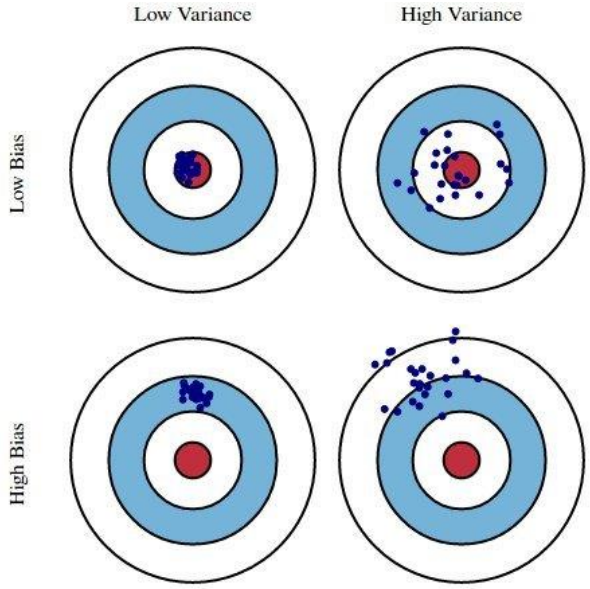
1. Create a list of lambdas (i.e. λ∈{0,0.01,0.02,0.04,0.08,0.16,0.32,0.64,1.28,2.56,5.12,10.24});
2. Create a set of models with different degrees or any other variants.
3. Iterate through the λs and for each λ go through all the models to learn some Θ.
4. Compute the cross validation error using the learned Θ (computed with λ) on the $J_{CV}(\Theta)$ **without** regularization or λ = 0.
5. Select the best combo that produces the lowest error on the cross validation set.
6. Using the best combo Θ and λ, apply it on $J_{test}(\Theta)$ to see if it has a good generalization of the problem.

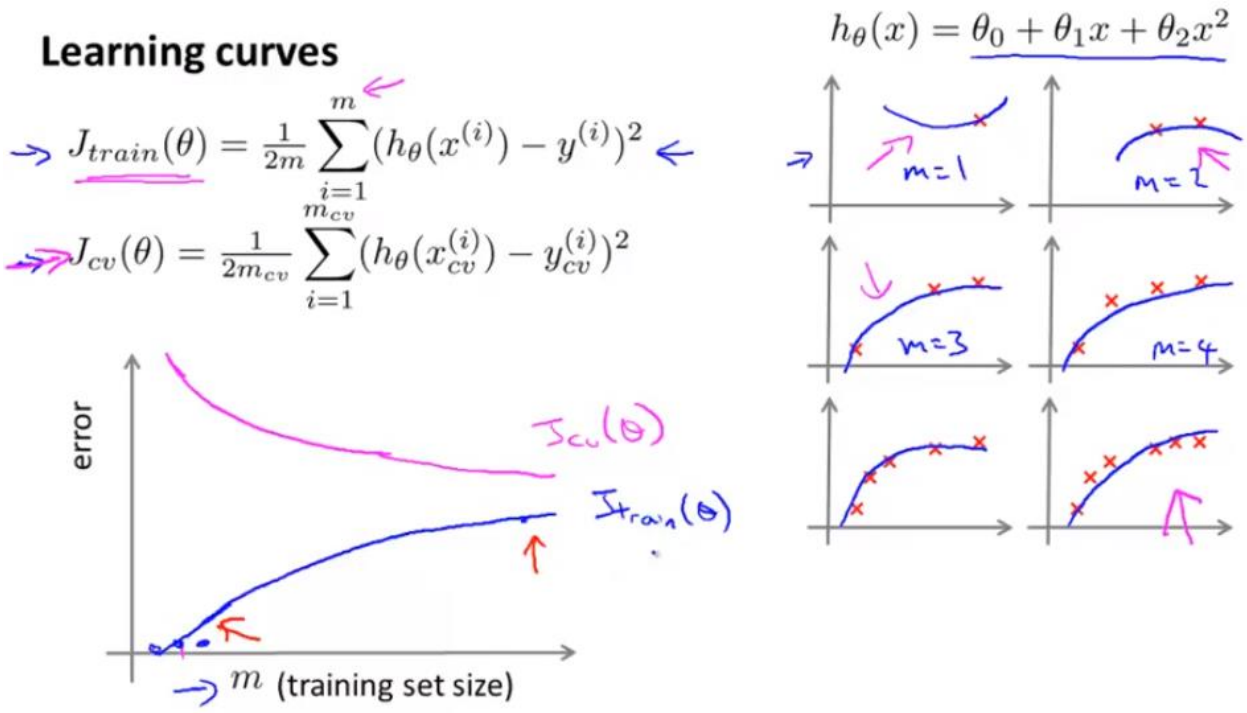
6.5. training set을 더 늘리는 것이 능사인지 알려면! Learning curve를 그려야 한다.

- Learning curve : code/알고리즘이 잘 작동하는지, bias나 variance가 있는지를 전반적으로 보는 도구이다.
- x축은 training set 표본의 개수.

(1) Training error : training set m이 작으면 fitting이 쉬우니까, training error가 작을 것이다. 그러나 m이 커질수록 training error가 커질 것이다.

(2) Cross validation error : m이 작으면 hypothesis가 좋지 않다. 그러나 m이 커지면 hypothesis가 generalize가 잘 된다. training이 잘 된다.





6.5.1. high bias의 경우

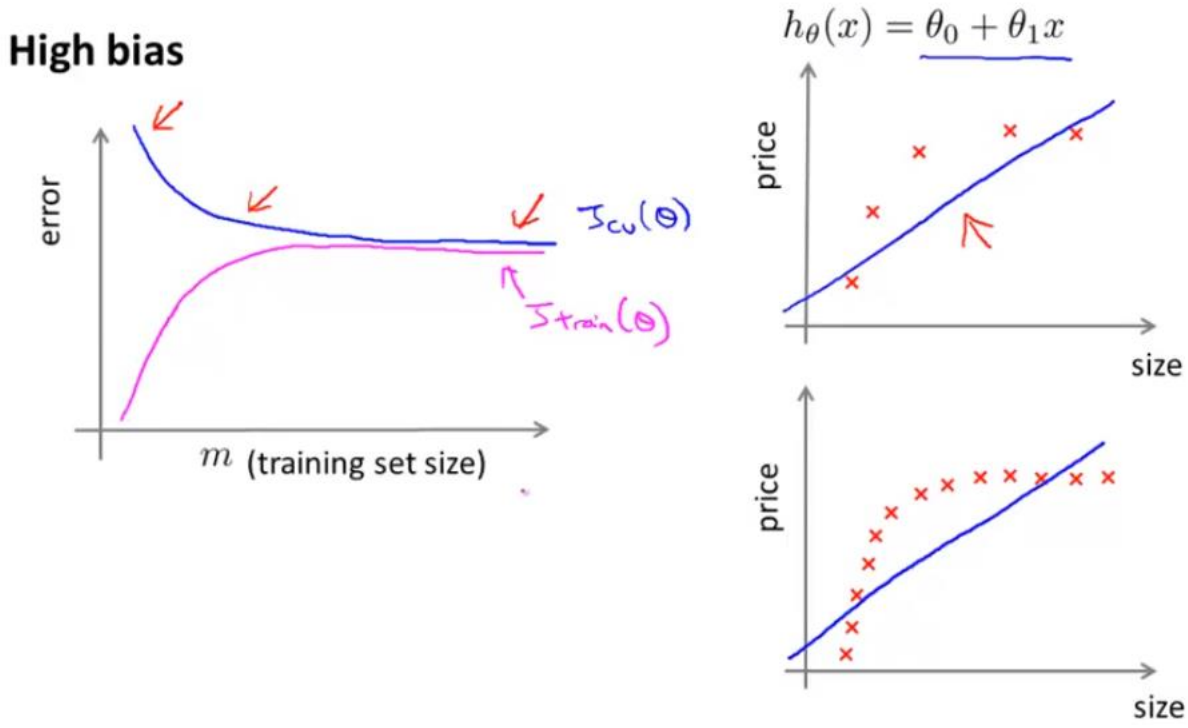
(1) Validation error : training set 개수 m이 커진다고 해서 회귀직선이 크게 개선되지는 않는다. m이 커진다고 해도 직선일 뿐이다.

→ 흐.. 이 validation error를 어떻게 정의하는 것인가? 무엇을 validation하는 것인가? 아 training set의 수에 따라서 그것을 validation하는구나.

(2) Training error : m이 작을 때는 fitting이 잘 되므로 error가 작지면 m이 커질수록 error가 커지게 되는데 계속 커지지는 않는다. 아 training error와 validation error가 수렵하는데 이건 당연한 현상인건가???

→ 질문하기.

(3) 결론 : High bias일 때는 Validation error랑 Training error가 둘 다 크다. Bias가 크기 때문에(parameter가 2개 밖에 없어) 그리고 training set을 더 늘린다고 해서 딱히 error가 개선되지 않는다.

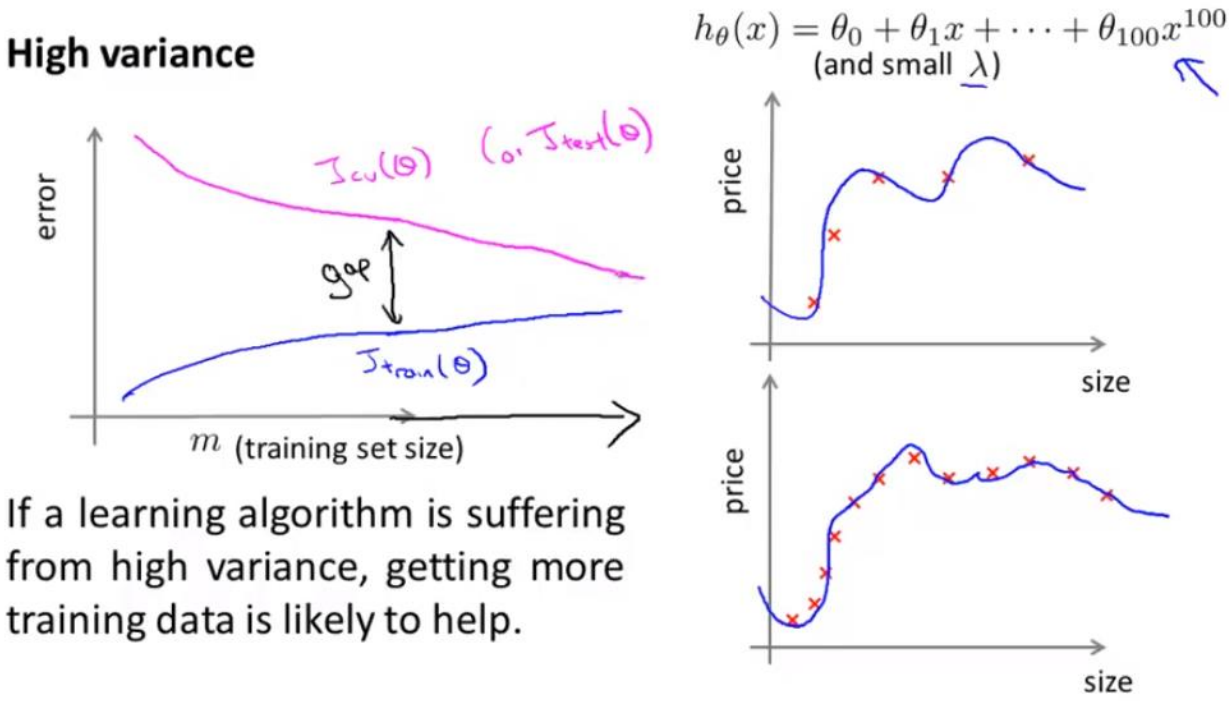


6.5.2 High variance의 경우

(1) Training error : 표본이 많아지면, overfitting을 하긴 하겠지만 fitting을 모든 점에 하기 힘들어지기 때문에 training error가 조금 커진다. 그래도 낮은 편

(2) Validation error : overfitting이니까 일단 error는 높은 편. m이 커지면 그나마 조금 error가 줄어든다.

(3) 결론 : High variance의 경우에는 gap이 있다. training example을 늘리면 cross validation error을 줄일 수 있겠구나! 도움이 된다.



- 정리 :

Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

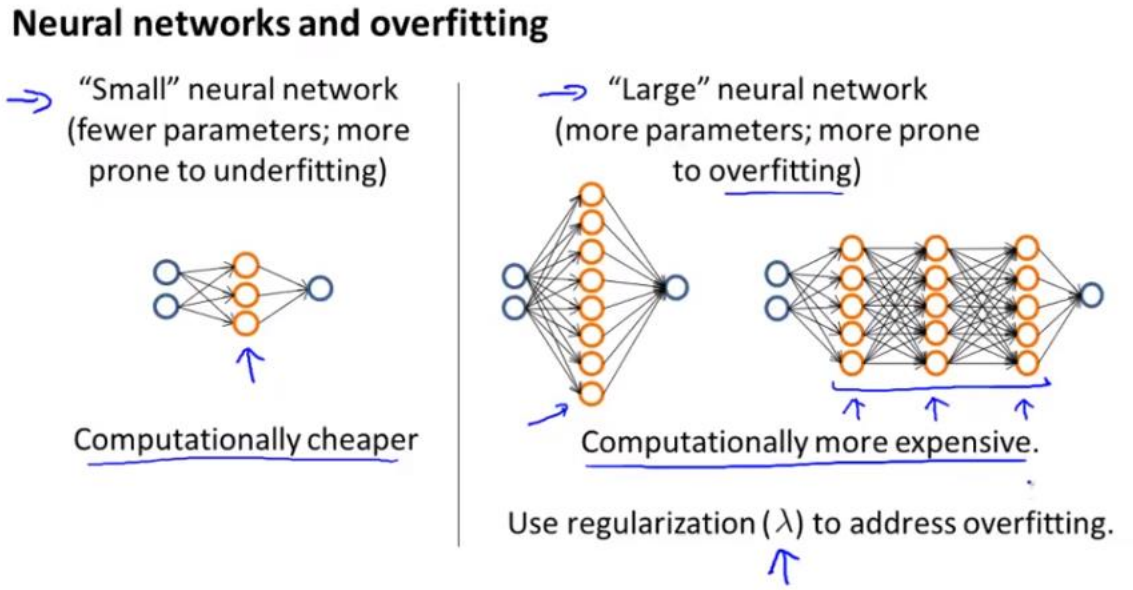
5.6. Neural Network의 경우에 고려사항

(1) hidden layer의 개수 :

- 너무 많으면 overfitting이 된다. → regularization 사용
- A neural network with fewer parameters is prone to underfitting. It is also computationally cheaper.
- A large neural network with more parameters is prone to overfitting. It is also computationally expensive. In this case you can use regularization (increase λ) to address the overfitting.

→ 질문!! 왜 layer가 많으면 parameter가 많은 효과가 나는 것일까?

→ parameter가 많아지니까. 행렬로서!



- hidden layer를 1개 말고 다른 거를 정하고 싶을 때! / hidden layer 1개가 default이다!

: hidden layer 2개, 3개, 4개 등을 해보고 cross validation set, test set으로 어떤 것이 가장 잘 되는지 확인해서 정해본다.

7. Machine learning system design : Building a spam classifier

- 핵심 : 어떻게 learning algorithm을 평가하고 개선할 것인가?
- 어떻게 접근해야 할까? Prioritizing what to work on ← gut feeling으로 이 중에서 하나를 고르지 말 것.

How to spend your time to make it have low error?

- Collect lots of data
  - E.g. "honeypot" project.
- Develop sophisticated features based on email routing information (from email header).
- Develop sophisticated features for message body, e.g. should "discount" and "discounts" be treated as the same word? How about "deal" and "Dealer"? Features about punctuation?
- Develop sophisticated algorithm to detect misspellings (e.g. m0rtgage, med1cine, w4tches.)

So how could you spend your time to improve the accuracy of this classifier?

- Collect lots of data (for example "honeypot" project but doesn't always work)
- Develop sophisticated features (for example: using email header data in spam emails)
- Develop algorithms to process your input in different ways (recognizing misspellings in spam).

→ 할 수 있는 가능성을 모두 열어두면 좋을 것 같다!

- 해결방법 : Recommended approach

(1) start a simple algorithm that you can implement quickly : 일단 빨리 만들어서 해보자!!

(2) **Learning curve** 그려서 high variance인지 high bias인지 살피기 → more data helpful? more features helpful?

(3) Error analysis : Manually examine the examples that your algorithm made errors on.

→ 1) what type of email it is → 수동 작업으로 쭉 보는 것이 중요하다.

Error Analysis

m_{CV} = 500 examples in cross validation set
Algorithm misclassifies 100 emails.
Manually examine the 100 errors, and categorize them based on:
→ (i) What type of email it is — pharma, replica, steal passwords, ...
→ (ii) What cues (features) you think would have helped the algorithm classify them correctly.

Pharma: 12
Replica/fake: 4
→ Steal passwords: 53
Other: 31

→ Deliberate misspellings: (m0rgage, med1cine, etc.)
→ Unusual email routing:
→ Unusual (spamming) punctuation:

7.1. Error analysis

7.1.1. Numerical evaluation (Error matrix)

→ 하나의 single real number로 평가하는 방법 ex) error rate

Should discount/discounts/discounted/discounting be treated as the same word?
Can use "stemming" software (E.g. "Porter stemmer")
universe/university.
Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.

→ stemming을 한 것과 stemming을 안 한 것을 cross validation set로 error rate를 본다.

Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.
Without stemming: 5% error With stemming: 3% error
Distinguish upper vs. lower case (Mom/mom): 3.2%

7.1.2. Evaluation for Skewed classes (classification)

- 정의 : skewed classes에서는 위의 single real number error matrix를 쓸 수가 없다. 그 이유는 암 발병률과 같이 원채 확률이 낮은 경우에 learning algorithm 자체의 error rate와 비슷할 수 있기 때문이다. y=0으로 고정시킨 algorithm에서도 error rate가 0.5%이다. 하지만 이 알고리즘이 좋은 알고리즘은 아니다!! 따라서 다른 알고리즘을 사용할 필요가 있다.

Cancer classification example

Train logistic regression model $h_\theta(x)$. ($y = 1$ if cancer, $y = 0$ otherwise)

Find that you got 1% error on test set.

(99% correct diagnoses)

Only 0.50% of patients have cancer.

→ skewed classes.

```
function y = predictCancer(x)
  y = 0; %ignore x!
return
```

→ 0.5% error

→ 99.2% accuracy (0.8% error)

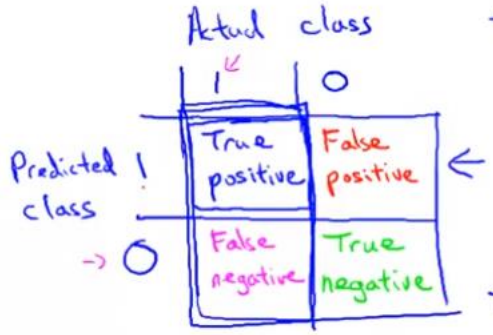
→ 99.5% accuracy (0.5% error)

→ test set의 error가 1%인데, 발병률이 0.5%이면 확인하기가 어렵지 않겠느냐. test set에서 error가 나는 이유는 대표성이 저하될 수 있기 때문이다.

- 해결법 : Precision/Recall
  - 아까와는 달리 암에 걸린 것을 사건 1로 둔다.
  - Precision의 정의는 predicted positive 중에서 true positive의 수를 세는 것이다.
  - Recall의 정의는 진짜 암에 걸린 사람들 중에서 암으로 정확하게 진단한 경우(true positive)를 세는 것이다.

Precision/Recall

$y = 1$ in presence of rare class that we want to detect



→ **Precision**

(Of all patients where we predicted $y = 1$, what fraction actually has cancer?)

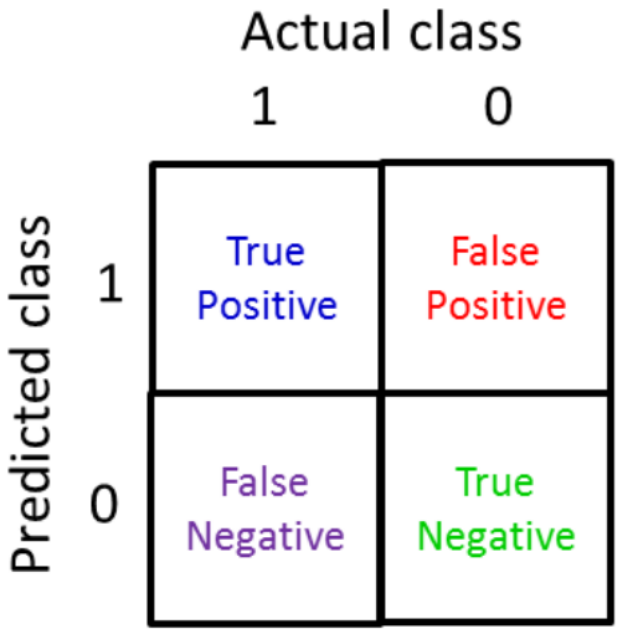
$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positive}}{\text{True pos + False pos}}$$

→ **Recall**

(Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$\frac{\text{True positives}}{\#\text{actual positives}} = \frac{\text{True positives}}{\text{True pos + False neg}}$$

→ recall을 사용하게 되면 위에서 y=0만을 predict하는 알고리즘을 걸러낼 수 있다. 왜냐하면 진짜 암환자 중에서 암이 있다고 진단을 한 비율이 0이기 때문이다.



$$\text{Precision} = \frac{\text{True positives}}{\#\text{ predicted as positive}} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

$$\text{Recall} = \frac{\text{True positives}}{\#\text{ actual positives}} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

- precision and recall의 조절 방법
  - modify the classification algorithm's threshold to gain higher precision or recall
  - y = 1, y =0을 정하는 기준값을 변경하면 이를 가능하게 할 수 있다.

Logistic regression: $0 \le h_\theta(x) \le 1$
Predict 1 if $h_\theta(x) \ge 0.5$ 0.7 0.9
Predict 0 if $h_\theta(x) < 0.5$ 0.7 0.9

(1) Precision을 높이는 방법 :

threshold를 높인다. 그러면 분모가 줄어든다. 암환자로 진단된 환자가 줄어드니까. 정확도가 높아지는 것이다.

Suppose we want to predict $y = 1$ (cancer) only if very confident.
→ Higher precision, lower recall.

Suppose we want to avoid missing too many cases of cancer (avoid false negatives).
→ Higher recall, lower precision.

(2) Recall을 높이는 방법 :

threshold를 낮춘다. 그러면 분자 true positive는 높일 수 있다. false negative를 낮출 수 있다.

- threshold를 정하는 방법
  - precision, recall의 tradeoff가 있는데 어떤 값을 가진 algorithm을 사용하는 게 나을까? single number evaluation matrix로 만들면 바로 바로 판단할 수 있다.

F_1 Score (F score)

How to compare precision/recall numbers?

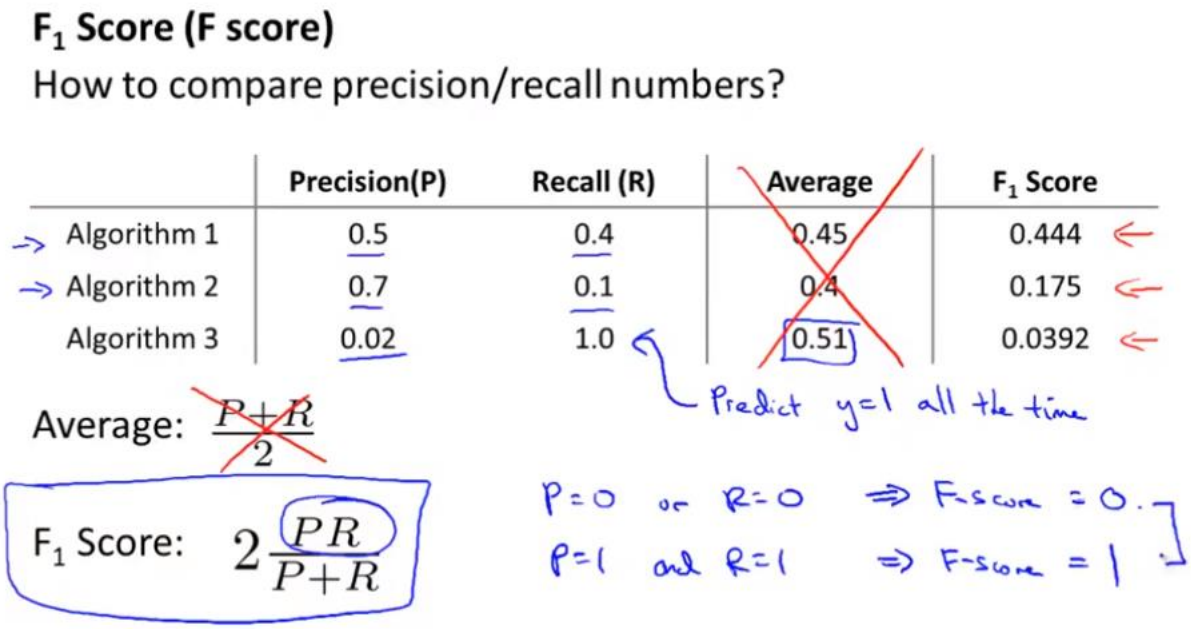
	Precision(P)	Recall (R)
Algorithm 1	0.5	0.4
Algorithm 2	0.7	0.1
Algorithm 3	0.02	1.0

- Threshold 고르는 방법

- Sol1 : 평균을 구해보자. 평균이 좋은 single number matrix가 될 수 있을까? 아니다. y=1만 예측하는 알고리즘을 해보면 precision이 낮고, recall이 높은 algorithm 3과 같은 결과가 나오는데 이 결과가 average가 가장 높은 것을 보면 별로 좋지 못한 방법이라는 것을 알 수 있다.

- Sol2 : F1 Score (2 * PR / P + R) → 이 방법을 사용하면 0에 가까우면 가중치를 높게 주기 때문에 P나 R이 0에 가까우면 F score도 0에 가까워진다. 따라서 P나 R이 0이나 1로 극단적인 경우를 걸러낼 수 있다.

→ cross validation set을 통해서 F1 score를 maximization 할 수 있는 threshold를 고르면 된다.



7.2. Large data for machine learning

how much data to train on.

- 어떤 조건에서 많은 데이터가 도움이 될까?

- 예시 : logistic regression을 했을 때, training set size가 큰 것이 좋다라는 결론을 낸 연구

Designing a high accuracy learning system

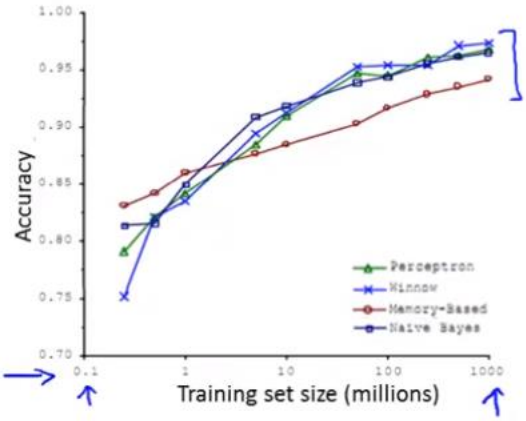
E.g. Classify between confusable words.

{to, two, too} {then, than}

→ For breakfast I ate two eggs.

Algorithms

- → - Perceptron (Logistic regression)
- → - Winnow
- → - Memory-based
- → - Naïve Bayes



"It's not who has the best algorithm that wins.
It's who has the most data."

- Large data가 효과가 있을 조건

(1) Many parameter!!!! feature x (R^{n+1})가 y를 예측하기 위한 충분한 정보가 있다는 전제가 필요하다.

Example: For breakfast I ate two eggs. ←

Counterexample: Predict housing price from only size (feet^2) and no other features. ←

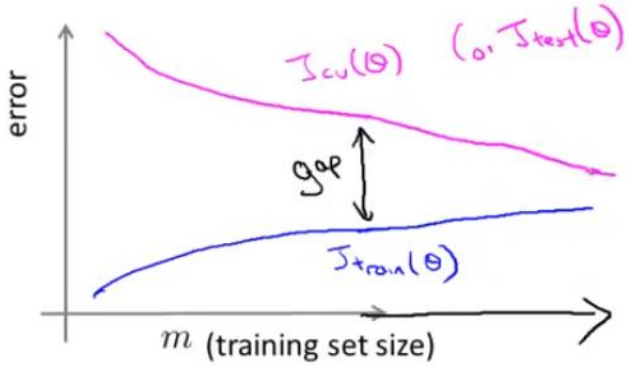
→ 뭐가 data가 충분한 것인지 어떻게 아는가? : **human expert한테 물어봤을 때, y를 잘 예측할 수 있으면 정보가 충분한 것이다.**

- 정보가 많다는 것은 무엇일까? parameter가 많다는 뜻이다.

Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

<reminder>

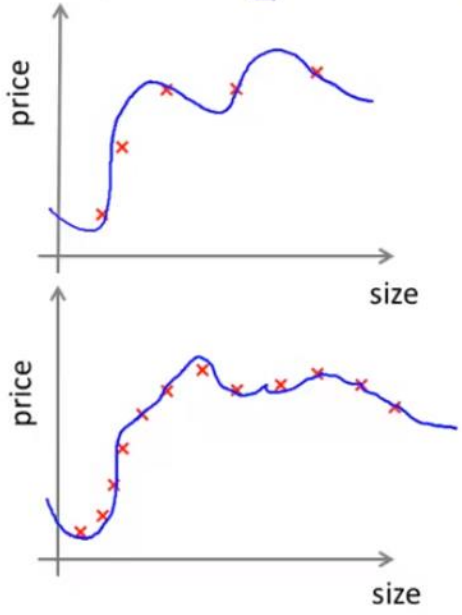
High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help.

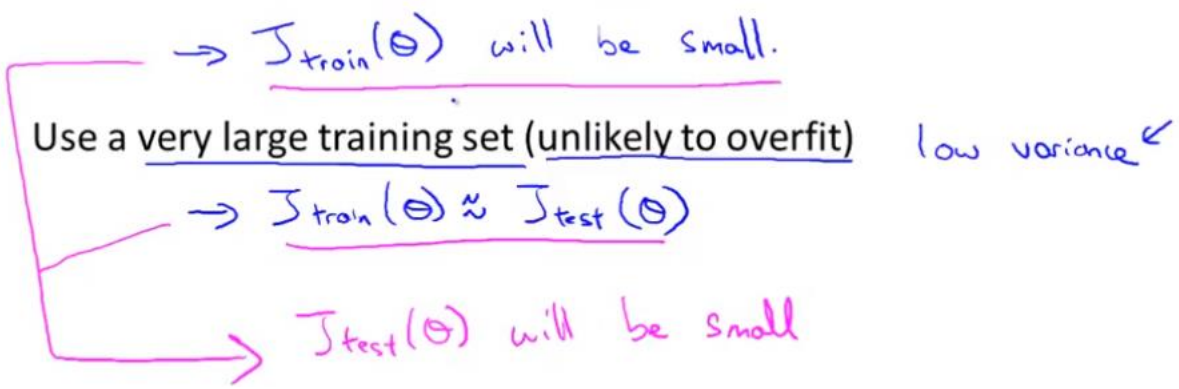
$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$

(and small λ)



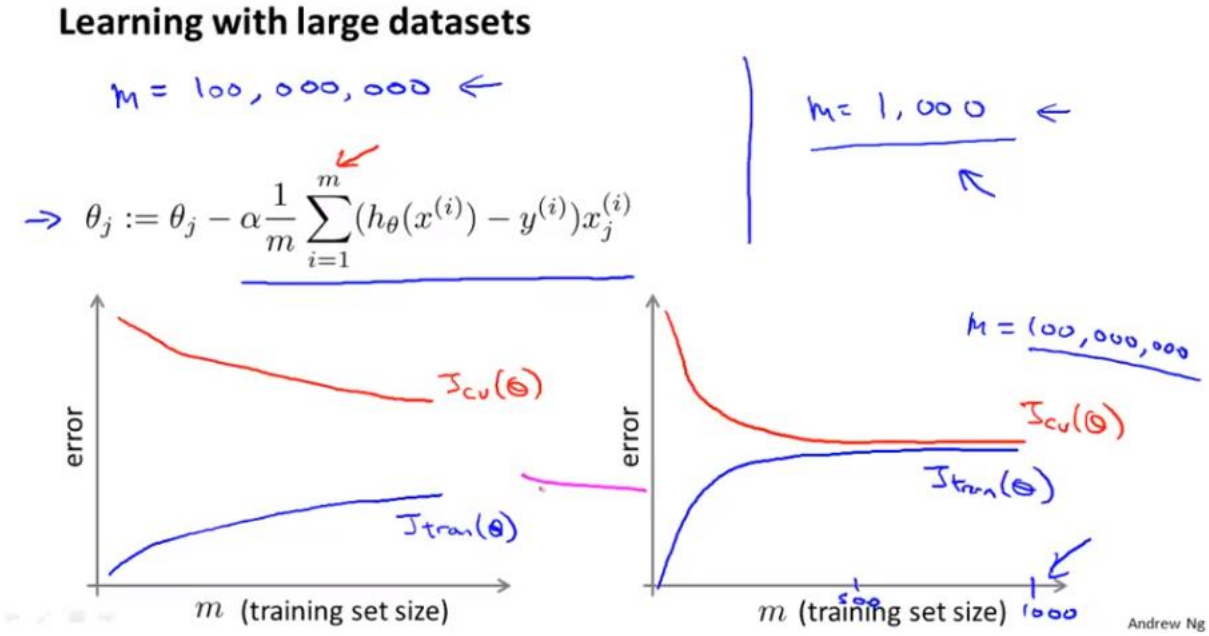
→ 그러나 overfitting문제가 발생할 가능성도 있다. 따라서 (2)번 조건이 필요한 것이다.

(2) Use very large training set → unlikely to overfit



(1) (2)조건이 다 갖춰지면 large data를 가졌을 때 high performance learning algorithm에 도달할 수 있다.

- 정리 : training data set(m) = 1억개가 있다고 하더라도 1000까지 해본다. error curve를 그려보고 high varianc양상을 보인다면(왼쪽 그래프) data set을 많이 하는 것이 좋겠다는 결론을 내릴 수 있게 된다. 만약 오른쪽 그래프처럼 나온다면 feature를 더하거나, neural network의 hidden layer를 더하거나 해서 그래프가 왼쪽과 같이 나오도록 할 수 있다.



7.2.1. large data set으로 training을 할 때, 알고리즘을 효율화할 수 있는 3가지 방법

(1) Stochastic gradient descent

(2) mini-batch gradient descent

(3) Map reduce and data parallelism

(1) Stochastic gradient descent

- 사용 목적 : 기존에 우리가 알고리즘을 만드는 방법은 우선 minimize를 하고 싶은 cost function을 찾고 그것을 minimize할 수 있는 parameter를 찾는 것이었다. 그 때, gradient descent를 썼었다. 하지만 이는 computationaly expensive해질 수 있다.
- 예시 : linear regression을 예로 사용하지만 logistic regression, neural network에서도 가능하다.

Linear regression with gradient descent

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

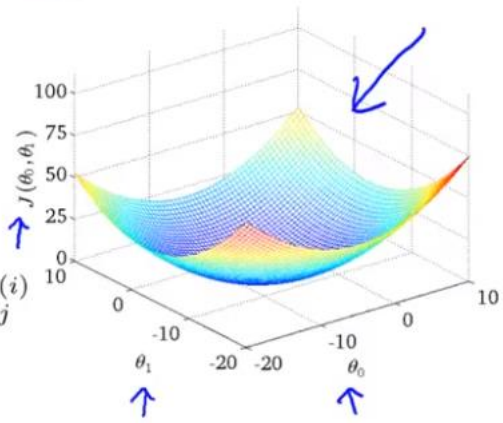
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



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

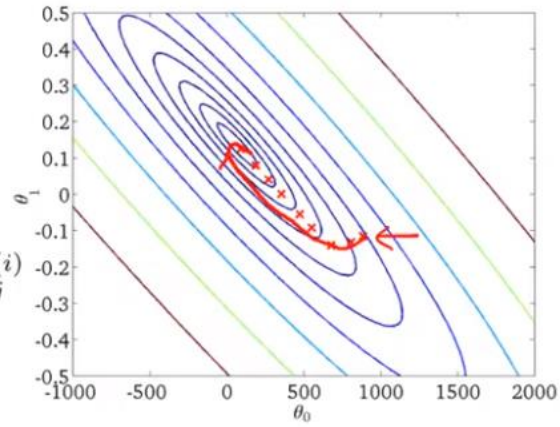
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

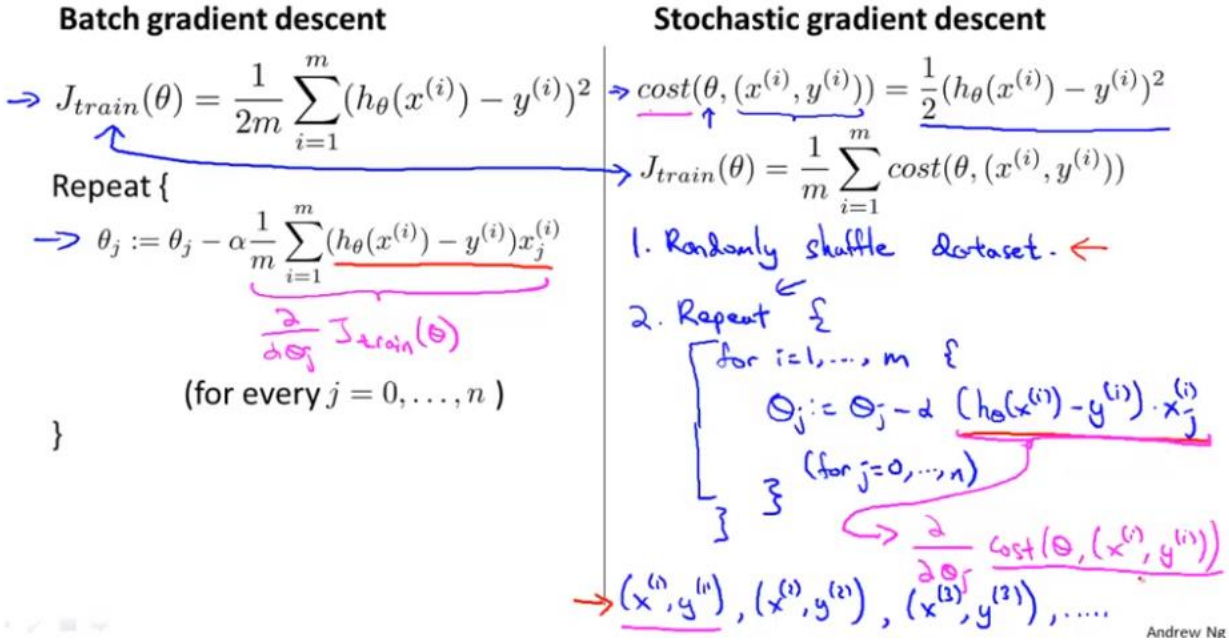
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

}



- 문제 설정 : batch gradient descent이라고 부른다.(기존에 우리가 배운 것) 근데 m = 3억 정도 되면 3억을 메모리에 넣고 한번 gradient descent를 하고 다시 3억개를 메모리에 넣고 더해서 gradient descent 한 발짝가고 해야 한다. 이는 비효율적이다.
- 문제 해결 : m개 data set을 다 더해서 빼서 parameter를 update하는게 아니라, 만약에 data set 1개씩해서 update를 하게 되면 어떻게 될까?



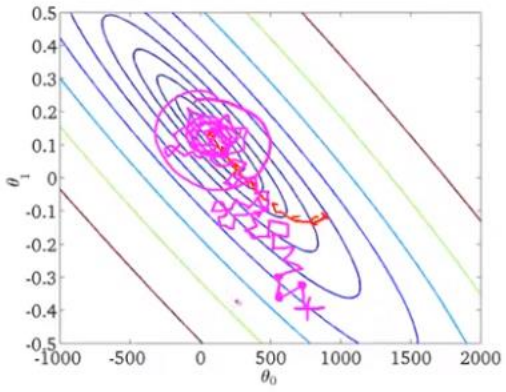
- 정리 : 1개씩 하더라도 batch gradient descent로 했을 때와 가까운 parameter를 얻을 수 있게 된다. m이 엄청 크다면 1번 ~ 10번 정도만 하면 된다.

→ 질문! 정말 그런가?? 너무 적은거 아닌가? 왜 m이 큰 경우에는 이게 가능한 것인가?

Stochastic gradient descent

1. Randomly shuffle (reorder) training examples

2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for every $j = 0, \ldots, n$)
   }
   }



- 주의 사항

1) Checking for convergence : J cost function의 값이 converging하는지를 보아야 한다. stochasitc을 할 때는 그렇게 computing cost가 비싸지 않으므로 gradient descent를 하면서 바로 할 수 있다. cost를 ploting할 때 몇 개씩의 평균을 plotting할지를 정하고 보면 된다.

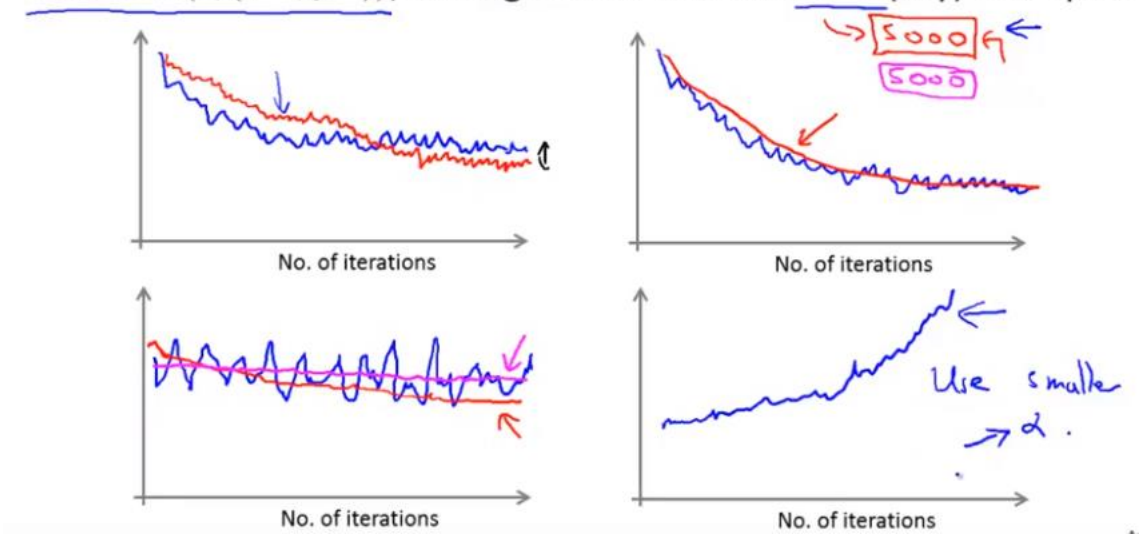
Checking for convergence

- Batch gradient descent:
  - Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.
  - $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ M = 300,000,000
- Stochastic gradient descent:
  - $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$ $(x^{(i)}, y^{(i)})$, $(x^{(i+1)}, y^{(i+1)})$, ...
  - During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.
  - Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

- cost를 구할 때 몇 개씩 평균해서 보여주는가에 따라서 learning curve가 달라진다. 만약에 1000개씩 평균하다가 5000개씩 평균하면 smoother graph가 나타난다.

Checking for convergence

Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples



2) learning rate의 조절

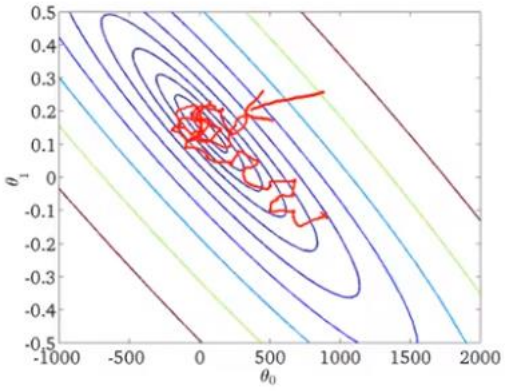
- learning rate(alpha)를 무엇을 쓰는가에 따라서도 graph 양상이 조금 달라질 수 있다.(learning rate이 작으면 cost가 떨어지는 것이 좀 더 느리다.)
- 만약에 diverging하면 alpha를 줄여보자.
- iteration이 증가할 때 learning rate를 줄이면 stochastic gradient descent가 마지막에 날뛰는 것을 줄일 수 있다.

Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.
2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }



Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{const1}{iterationNumber + const2}$)

Andrew Ng

(2) mini batch gradient descent ← stochastic와 batch 사이 (b=1, b=m사이)

- 목적 : iteration을 한번 할 때마다 1개만(stochastic)하는 게 아니라, batch를 2-100개씩 하는 것이다. 왜냐하면 vectorization하게 되면 훨씬 효율적으로 빠르게 계산할 수 있게 되기 때문이다. parellel하게 10개씩 할 수 있다. b = 10개로 잡는 것이 좋다.(Andrew 추천)

Mini-batch gradient descent

→ Batch gradient descent: Use all m examples in each iteration

→ Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

b = mini-batch size. b = 10. 2-100

Get b = 10 examples $(x^{(i)}, y^{(i)}), \dots (x^{(i+9)}, y^{(i+9)})$

$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$

$i := i + 10$

- 목적식 :

Mini-batch gradient descent

Say $b = 10, m = 1000.$

Repeat {

→ for $i = 1, 11, 21, 31, \dots, 991$ {

→ $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$

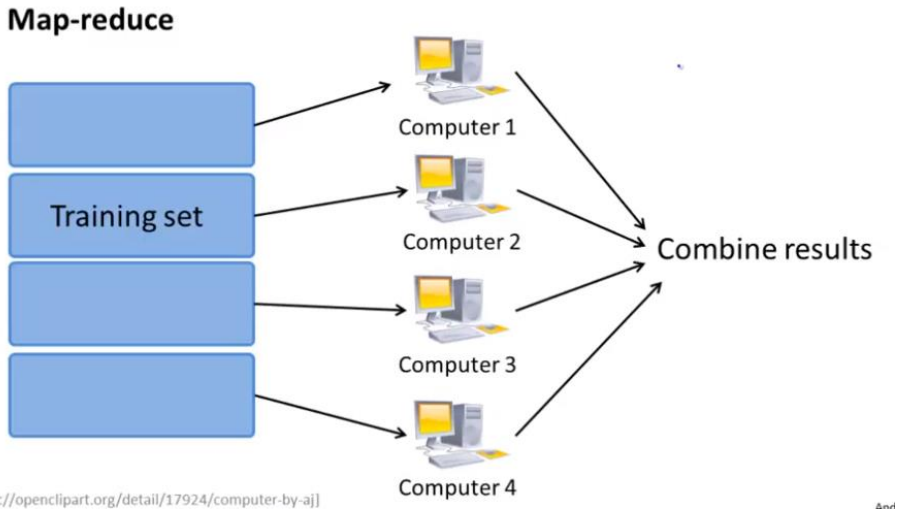
(for every $j = 0, \dots, n$)

}

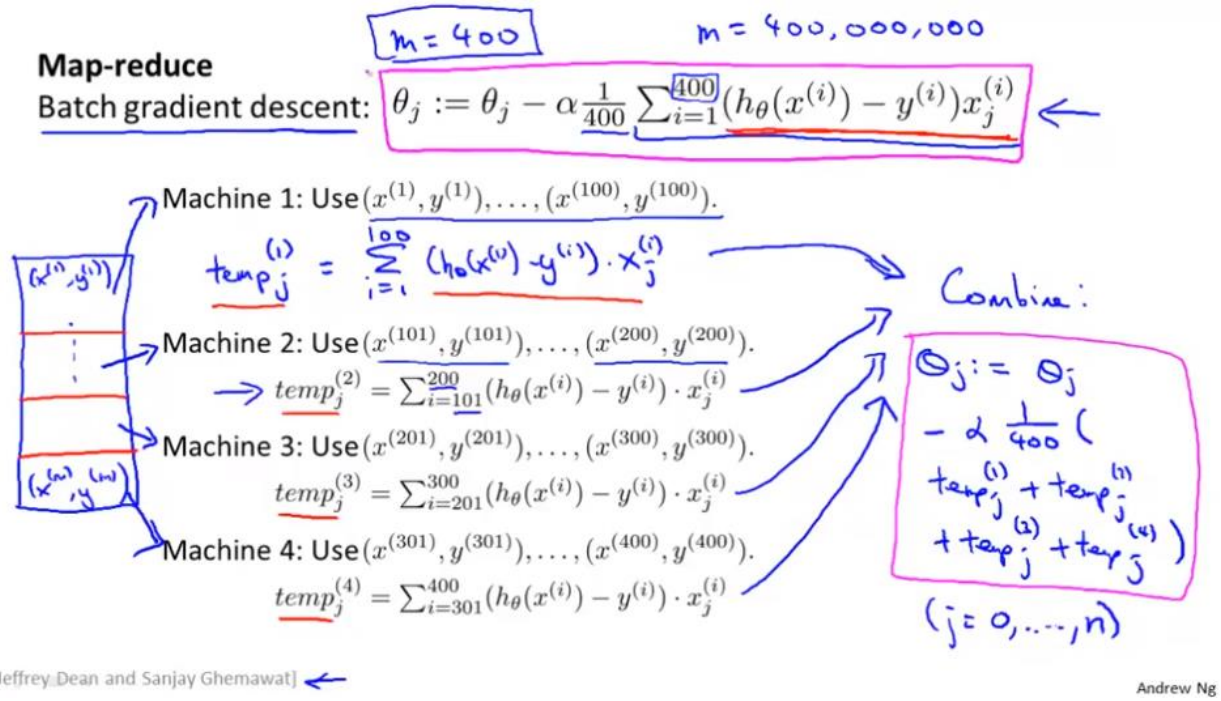
}

(3) Map reduce

- gradient descent만큼이나 효율화에 중요하다.
- 400개의 sum을 계산해야 하면 100, 100, 100, 100개로 나눠서 4개의 computer에서 나눠서 계산하면 더 빨라진다.



- 예시 :



- 한 개의 컴퓨터 : core가 여러 개인 computer에서 컴퓨터가 여러 개가 없어도 가능하다. data를 수신, 전송하는 시간도 적게 든다.(network latency) numerical linear library에서는 이것을 한 컴퓨터에서 저절로 해준다.

7.3. Large data generation (artificial data synthesis) → 없으면 만들면 되잖아!!!!

Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. "How much work would it be to get 10x as much data as we currently have?"
   - Artificial data synthesis
   - Collect/label it yourself
   - "Crowd source" (E.g. Amazon Mechanical Turk)

- **How much work would it take to get 10x much data?!!!** 이게 정말 좋은 질문이다!!!
- 직접 collect, label해도 그렇게 오래 걸리지 않는다면!
- artificial data synthesis → unlimited data for training data
  - font를 바꾸기

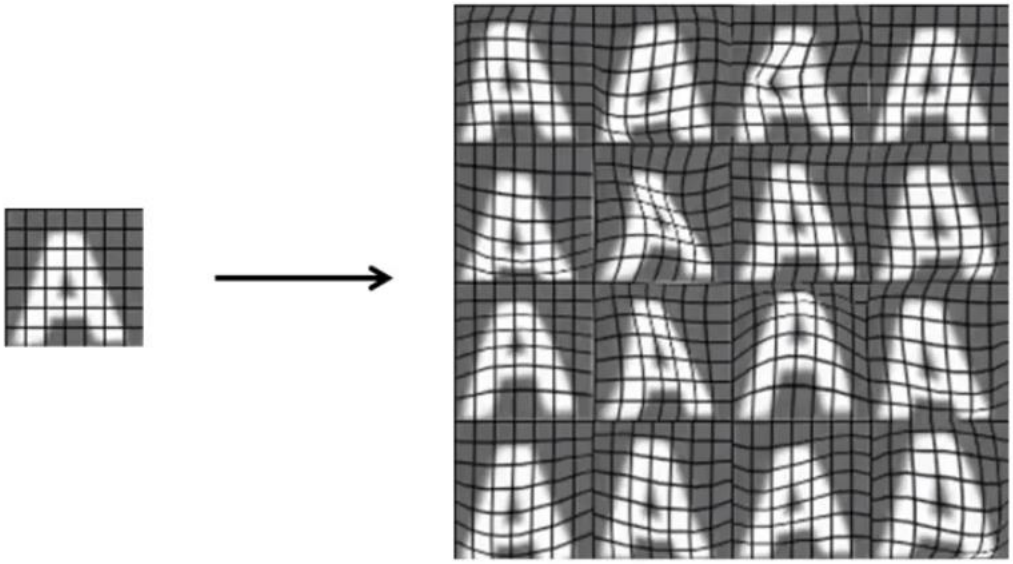
Artificial data synthesis for photo OCR

Real data

Synthetic data

- artificial warping

Synthesizing data by introducing distortions



- audio distortion

Synthesizing data by introducing distortions: Speech recognition

Original audio:

Audio on bad cellphone connection

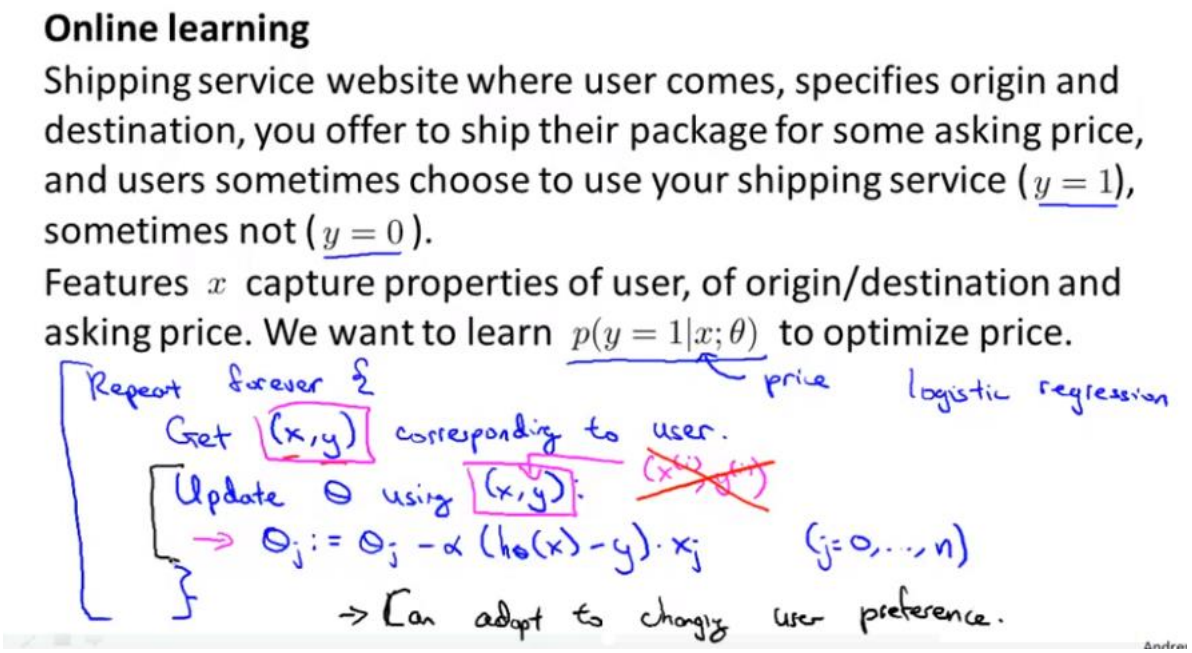
Noisy background: Crowd

Noisy background: Machinery

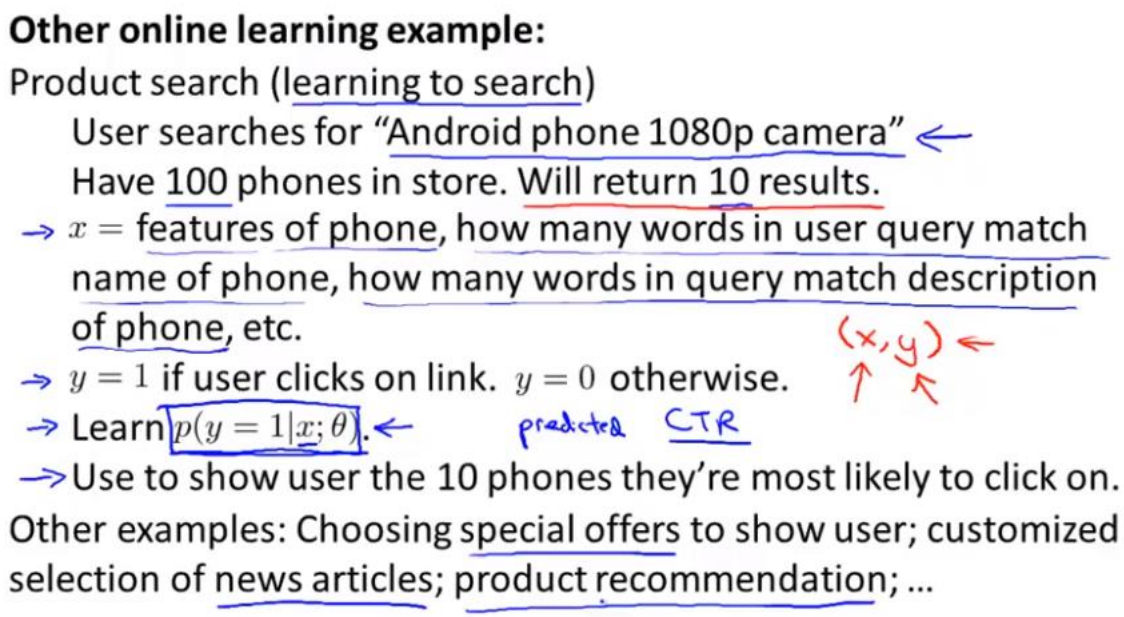
7.4. Online learning

- 목적 : online 상에서 데이터가 계속 들어오는 경우라면 learning을 계속 하게 하면 된다. 데이터를 얻고 그것을 통해서 learning하고, 그 data set을 버리면 된다. Small number of users라면 그것을 모아서 fixed learning set를 사용하면 된다.

  만약에 경제상황이 바뀌거나(탄력성이 바뀌거나) 하면 adapt to changing user preference를 할 수 있다.



- 예시 2 : user search query (predicted CTR click through rate)



8. Machine learning pipelines → Photo OCR(Optical character recognition) example

- 개요 : machine learning이 pipeline의 각 step에서 쓰이는 것이다.

Photo OCR pipeline

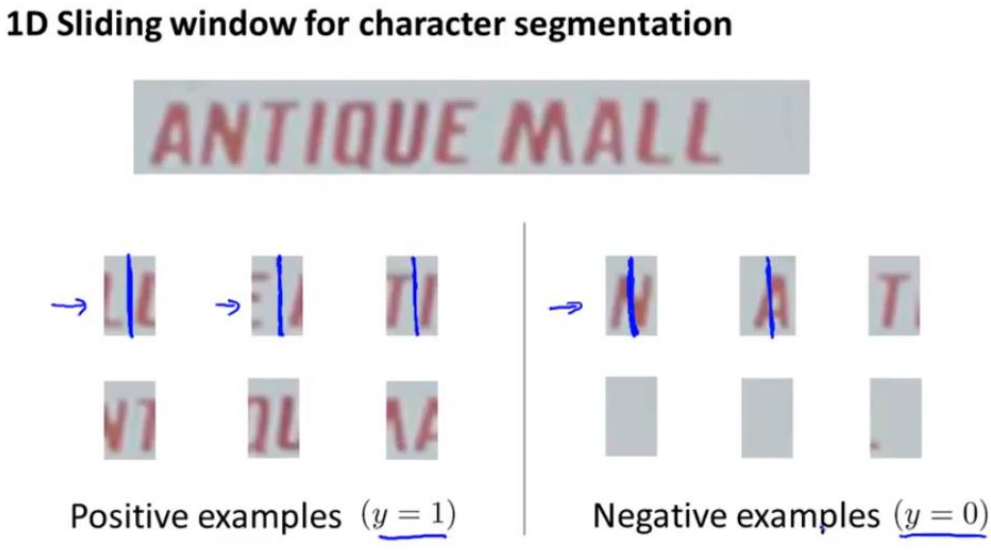
1. Text detection

2. Character segmentation

3. Character classification

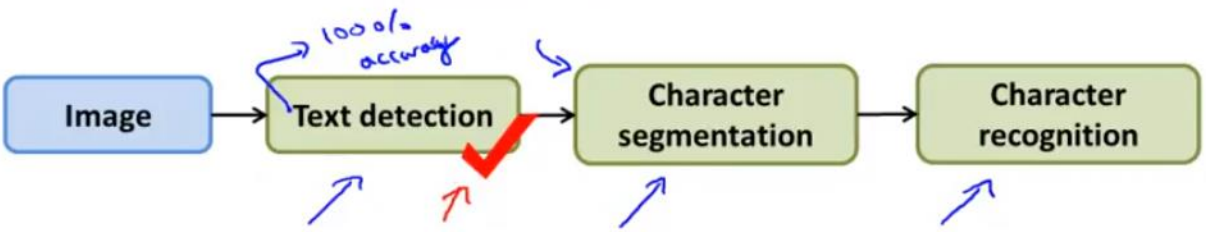
1) text detection

2) segmentation



8.1. ceiling analysis : 어떤 부분의 개발에 시간을 할당해야 하는지를 알려주는 분석

Estimating the errors due to each component (ceiling analysis)



What part of the pipeline should you spend the most time trying to improve?

Component	Accuracy
Overall system	72%
Text detection	89%

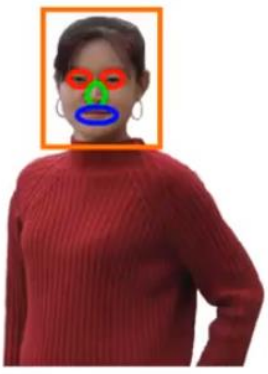
→ 여기서 체크된 부분을 인간이 스스로 해본다. 100% 정확할 때, accuracy가 어떻게 되는가? 72%에서 89%로 증가하는구나.

Component	Accuracy
Overall system	72%
Text detection	89%
Character segmentation	90%
Character recognition	100%

→ 아하! text detection 부분을 열심히 개선하면 되겠구나. 여기에 좀 더 engineer를 많이 투입하자.

Another ceiling analysis example

Face recognition from images
(Artificial example)



Component	Accuracy
Overall system	85%
Preprocess (remove background)	85.1%
Face detection	91%
Eyes segmentation	95%
Nose segmentation	96%
Mouth segmentation	97%
Logistic regression	100%

→ 여기서 중요한 것은 이 논리 구조이다. 이렇게 만약에 과학을 구획화할 수 있으면 어디에 집중할지를 좀 명확히 할 수 있지 않을까? mapping이 역시 중요한 것이다.

- don't trust your gut feeling!!!
  → systematic approach is better