Stanford course

topics covered and needs to practiced over python
types of algorithms : supervised , unsupervised
W1-W3
model & cost function
    contour plot
gradient descent
linear regression
batch gradient descent
multiple features
linear regression with multiple variables
vectorization of hypothesis function
gradient descent for multiple variable
feature scaling - gradient descent in practice I
    - mean normalization
learning rate - gradient Descent in practice II
Features and Polynomial Regression
    - Normal Equation
    - Normal Equation invertibility
W4
Logistic Regrssion
    Classification problems
binary classificaion , multiple classification
Hypothesis Representation
sigmoid/logistic function
interpretation of hypothesis output
decision boundary - linear and non-linear
Cost function in logistic regression model
convex optimization
optimatization algorithm - gradient descent (others : conjugate descent, BFGS, L-BFGS)
Cost function in logistic regression using gradient descent
Multiclass classification
one vs all
problem of overfitting, underfitting
regularization - cost function
    - intuition
regularized Linear regression

- gradient descent
- normal equation
- non-invertibility

regularized logistic regression
- cost function, gradient descent

advance optimization method for regularization

W5
neural Networks
nonlinear hypothesis
Non-linear classification
NN model representation I
- logistic unit, activation/sigmoid function

NN model representation II
- forward propagration - vectorized implementation
    - learning own features

non-linear decision boundary
multiclass classificaation based NN

Classification problem in NN
- cost function \\logistics regression

Backpropagation
- gradient computation
- algorithms
- intuition

Why backpropagation
advance optimization
learning algorithm
gradient checking
random initialization : symmetry breaking
training NN

W6
advice for applying ML
Deciding what to try next
debugging a learning algo
ML diagnostics
Evaluating hypothesis
Training/testing procedure for logistic regression
Model selection/training/validation/test sets

Bias & Variance
Error
Diaagnosing bais vs variance
effect of regularization on bias/variance
choosing regularization parameter
Bias/Variance as function of regularization parameter
Learning curves - high bias, high variance, error
NN and overfitting

Machine Learning system design
prioritizing what to work on : spam classifier
Error analysis
recommended approach
numerical evaluation of learning algo
handling skewed data
error metrics for skewed classes
Precison/Recall
Trading off precision and recall
Training large data sets
Designing a high accuracy learning system

W7

Explainations

transformers-Capsule Network

Week1

Week1
Machine Learning
A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

E = the experience of playing many games of checkers
T = the task of playing checkers.
P = the probability that the program will win the next game.

classified : supervised learning or unsupervised learning

Supervised Learning
we are given a dataset and already know what our correct output should look like,
having the idea that there is a relationship between the input-output
- regression
(predict results within a continuos output, map input variables to some continuos function)
- classification
(predict results in a discrete output, map input variables to discrete categories)
also called clustering algorithm

unsupervised learning
allows to approach problems with little or no idea what our results look like
derive structure from data where we don't know variable effects
derive this structure clustering based on reltionships among variables

Model and Cost Function

Linear regression
m = no of training examples
x's input var / features
y's output var / target variable
(x,y) - one training example
(x(i), y(i)) - ith training example

Training set

|

Learning Algorithms

|

X <- h - > Y

hypothesis : h function maps from X's to Y's

Y vs X graph, determine h(x) - Linear regressiom wit one variable(univariate linear regression)

h : X->Y so that h(c) is a good predictor for corresponding value of y

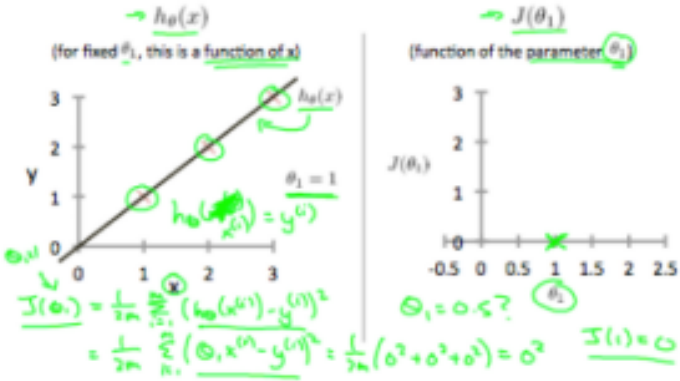
accuracy of the hypothesis function is measures by using cost function - average difference of all the results in hypothesis

J(O1,O2) = 1/2m (Summation (y'(i) - y(i))^2

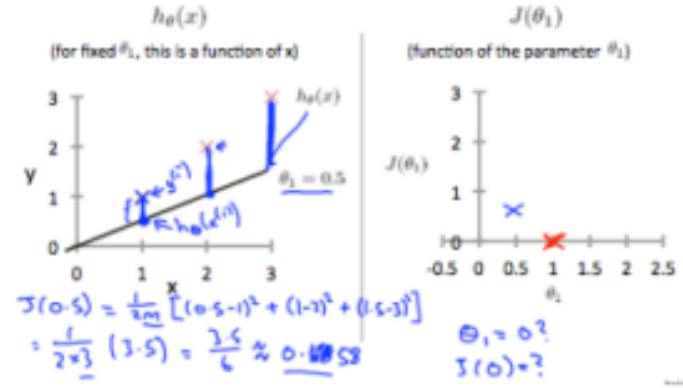
y'(i) = h(x(i))

minimize the difference between the predicted value and actual value (mean squared error)

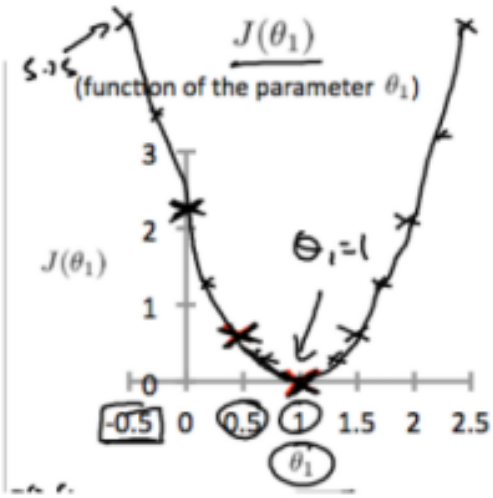
our objective is to get the best possible line so that avarge vertical distances of the scattered points from the line will be the least!



When $\theta_1 = 1$, we get a slope of 1 which goes through every single data point in our model. Conversely, when $\theta_1 = 0.5$, we see the vertical distance from our fit to the data points increase.



This increases our cost function to 0.58. Plotting several other points yields to the following graph:

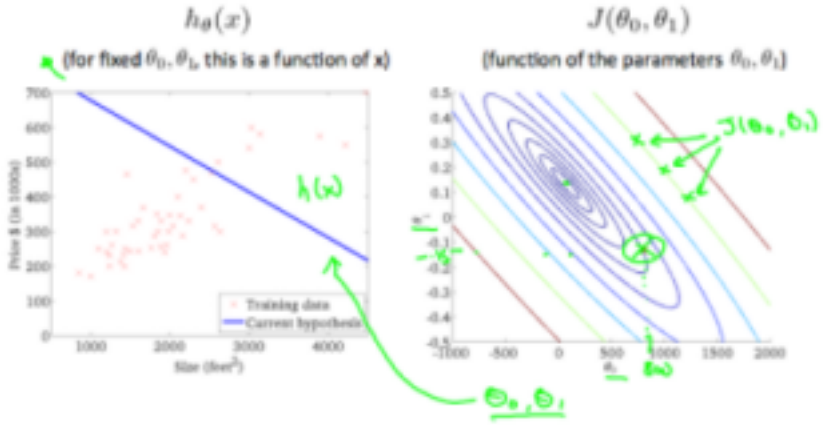


Thus as a goal, we should try to minimize the cost function. In this case, $\theta_1 = 1$ is our global minimum.

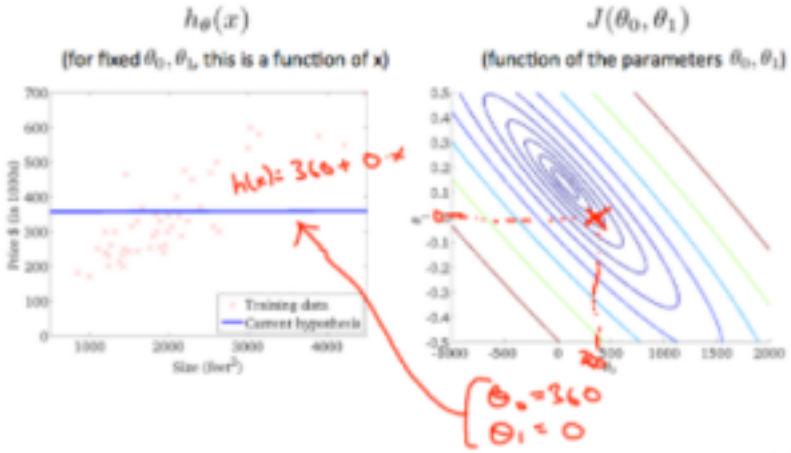
Contour plot - a graph that contains many contour lines

- it is a two variable function has a constanat value at all the points of the same line

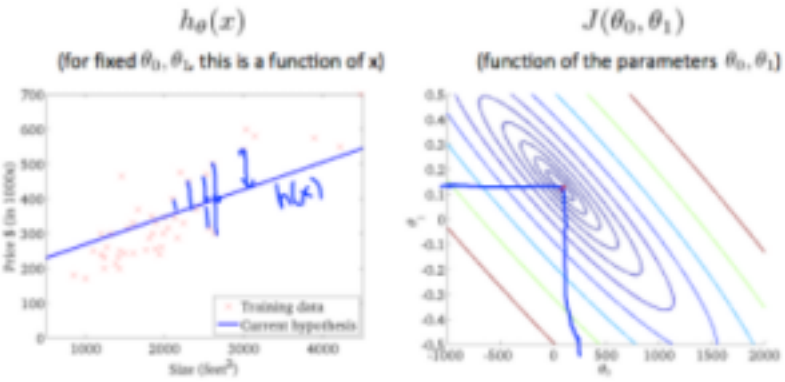
- same value of cost around consider single circle/ contour



Taking any color and going along the 'circle', one would expect to get the same value of the cost function. For example, the three green points found on the green line above have the same value for $J(\theta_0, \theta_1)$ and as a result, they are found along the same line. The circled x displays the value of the cost function for the graph on the left when θ_0 = 800 and θ_1 = -0.15. Taking another h(x) and plotting its contour plot, one gets the following graphs:

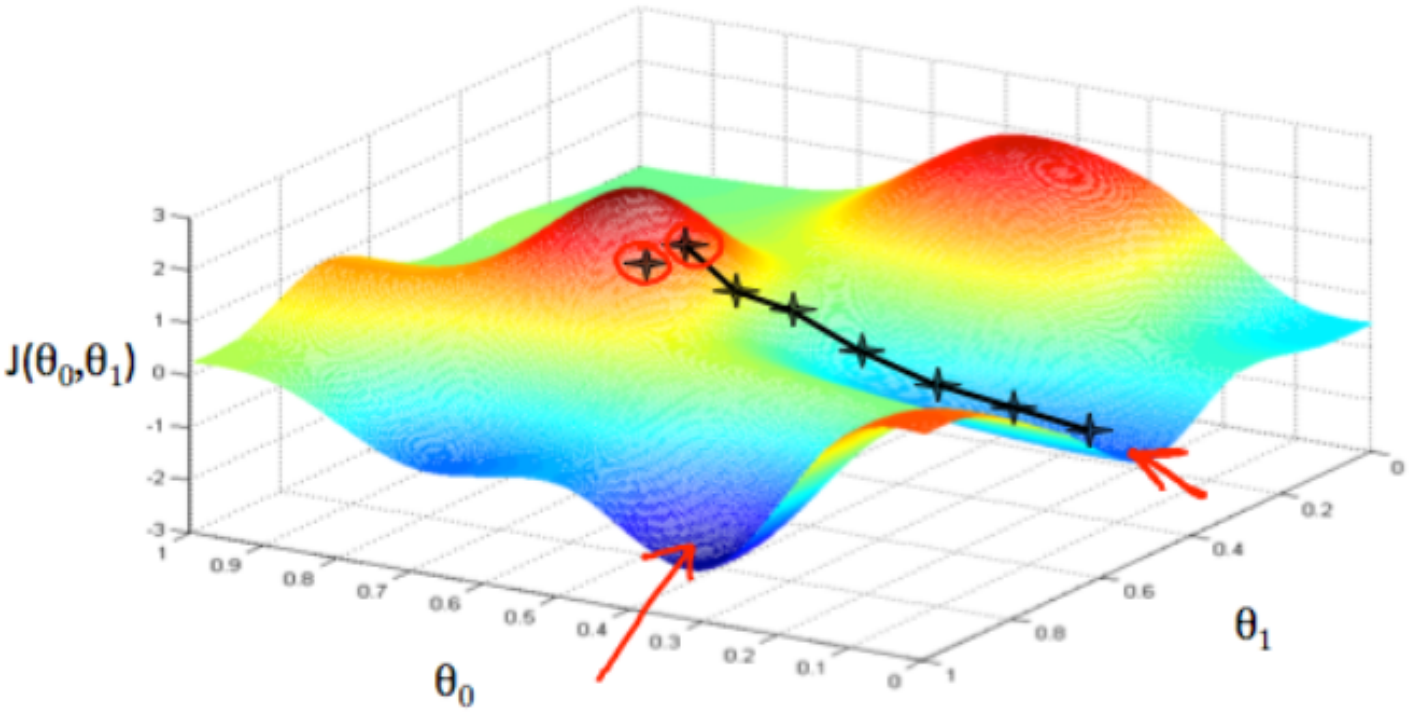


When θ_0 = 360 and θ_1 = 0, the value of $J(\theta_0, \theta_1)$ in the contour plot gets closer to the center thus reducing the cost function error. Now giving our hypothesis function a slightly positive slope results in a better fit of the data.



The graph above minimizes the cost function as much as possible and consequently, the result of θ_1 and θ_0 tend to be around 0.12 and 250 respectively. Plotting those values on our graph to the right seems to put our point in the center of the inner most 'circle'.

Gradient descent



we need to estimate the parameters in the hypothesis function
Red arrow shows the minimum points in the graph
based on the parameter range J(O1,O2), we find the cost function values
using a tangential line to a function(derivative of cost function), show increasing or decreasing condition, we follow this to direct towards the steepest descent.
size of each step is determined by the parameter (α alpha) called the learning rate
a smaller learning rate would be smaller step, and visa versa

direction is determined - using partial derivative of the J(O1,O2) until convergence

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

where

j=0,1 represents the feature index number.

At each iteration j, one should simultaneously update the parameters $\theta_1, \theta_2, ..., \theta_n$. Updating a specific parameter prior to calculating another one on the $j^{(th)}$ iteration would yield to a wrong implementation.

Correct: Simultaneous update

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

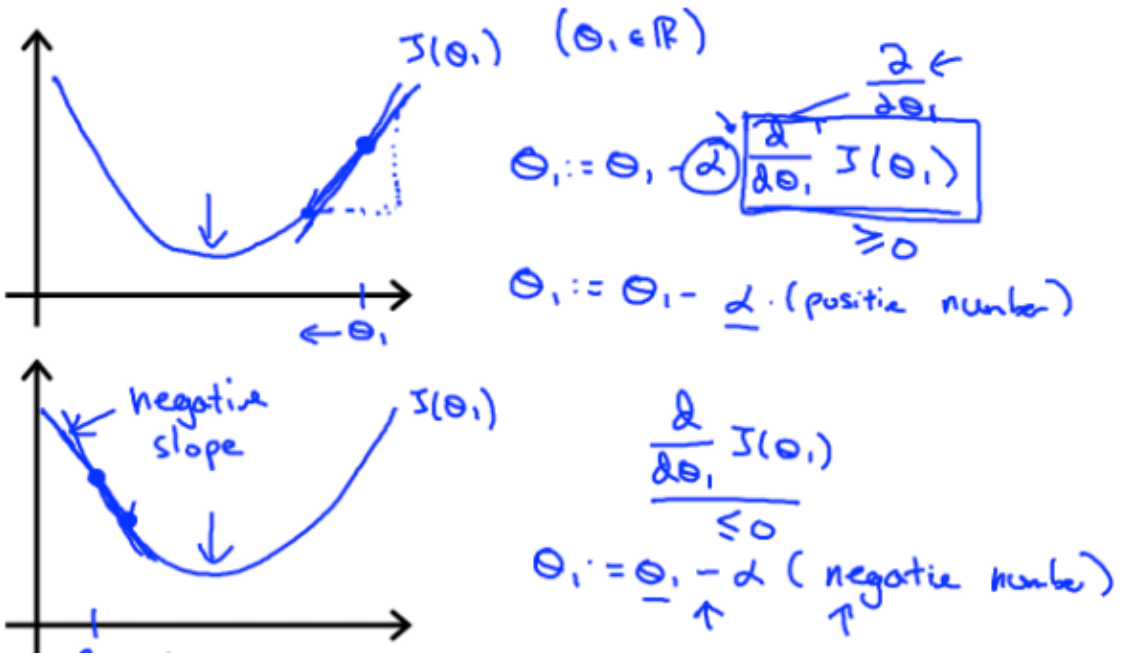
Incorrect:

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

$\theta 1 := \theta 1 - \alpha \, d\theta 1 \; dJ(\theta 1)$ - single parameter

More imporatantly

Regardless of the slope's sign for $\frac{d}{d\theta_1} J(\theta_1)$, θ_1 eventually converges to its minimum value. The following graph shows that when the slope is negative, the value of θ_1 increases and when it is positive, the value of θ_1 decreases.

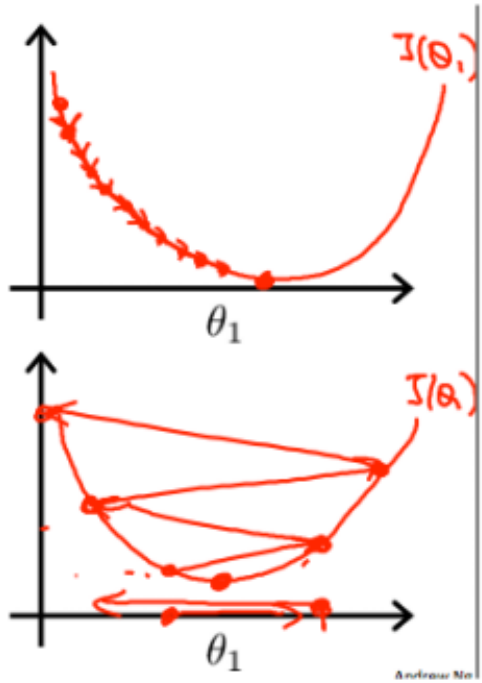


On a side note, we should adjust our parameter α to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value imply that our step size is wrong.

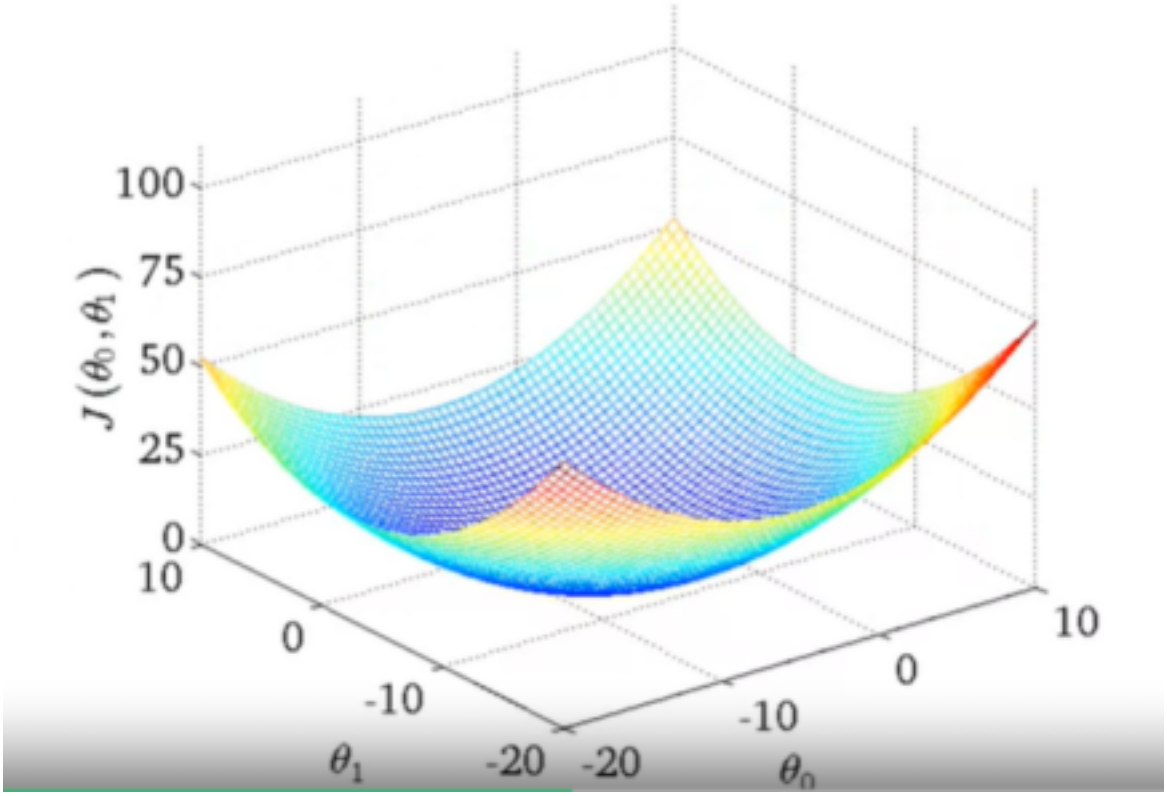
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

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.



The problem of gradient descent is different local optima (steepest/ lowest points) cost function of linear regression is always a bow shaped function :

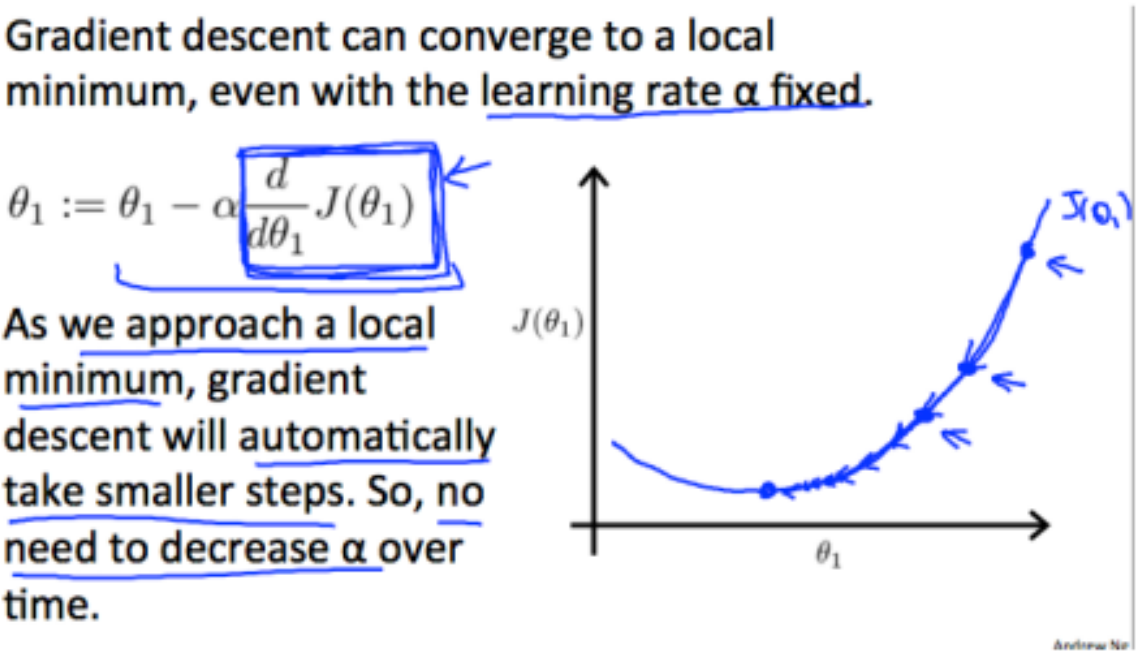


this is called convex function.
because of this, the problem of different local optima is not present, as it has single global optima

How does gradient descent converge with a fixed step size α?

The intuition behind the convergence is that $\frac{d}{d\theta_1}J(\theta_1)$ approaches 0 as we approach the bottom of our convex function. At the minimum, the derivative will always be 0 and thus we get:

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



applying gradient descent with linear regression
applying multivariable calculus

Gradient descent algorithm

repeat until convergence {

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

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

}

Linear Regression Model

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

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

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

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

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

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

repeat until convergence:

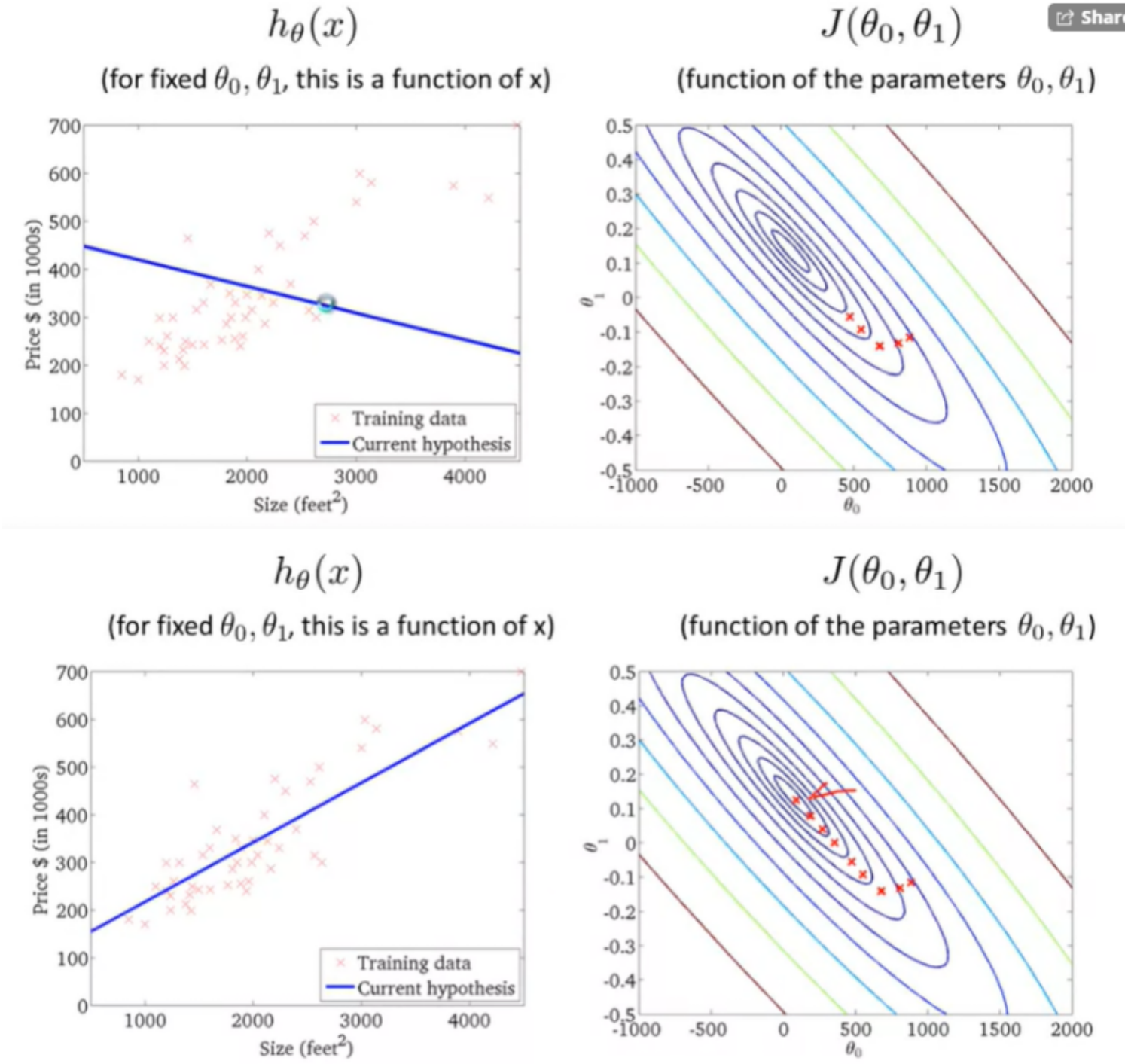
{

$\theta 0 := \theta 0 - \alpha 1/m \Sigma\, [i=1 \ldots m] \; (h\theta(xi) - yi)$

$\theta 1 := \theta 1 - \alpha 1/m \Sigma\, [i=1 \ldots m] \; ((h\theta(xi) - yi)\, xi)$

}

how the linear regression works if any point int he contour is consider and slowly it adjusted according to the center of the contour lines with h(x)



this is called batch gradient descent,

batch - each step of graident descent uses all the training examples

$\sum_{i=1...m} ((h_\theta(x_i) - y_i)\, x_i)$ this summation from 1 to m is what makes this called the batch

Week2

Multiple features (variables)

x1 x2 x3 x4 y

.

.

.

m times (m=47)

n :number of features (like size, no of bedrooms, no of floors, age of home)
x(i) :input (features) of ith training example
x(j,i) :value of feature of j in ith training example

hypothesis

h(x) = O1 + O2*x earlier
but now, h(x) = O1 + O2*x1 + O3*x2 + ...

Linear regression with multiple variables is also known as multivariate linear regression
x(*i*) =value of feature *j* in the *ith* training example
jx(*i*) =the input (features) of the *ith* training example
m =the number of training examples
n =the number of features

$h\theta(x)=\theta 0+\theta 1x1+\theta 2x2+\theta 3x3+\cdots+\theta nxn$

vectorization of our hypothesis function

gradient descent for multiple variable

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 $j = 0, \ldots, n$)

}

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

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

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

Feature scaling : gradient descent in practice I

idea : make sure features are on a similar scale

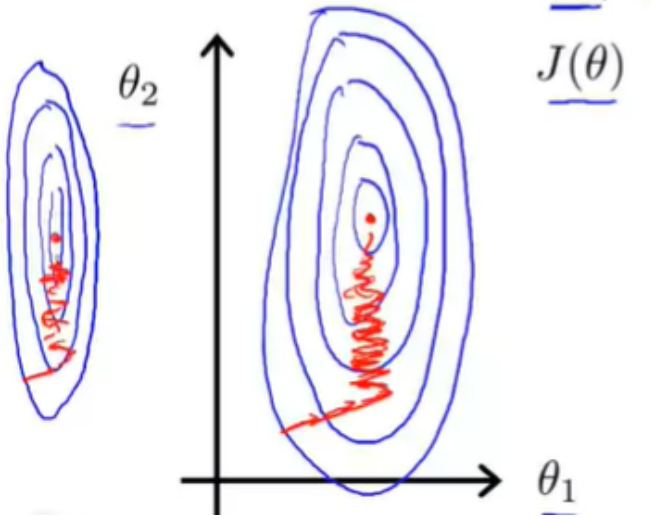
i.e x1 = size (0-2000)

x2 = no of bedrooms (1-5)

if we plot the contour of the cos function Jθ

E.g. x_1 = size (0-2000 feet^2)

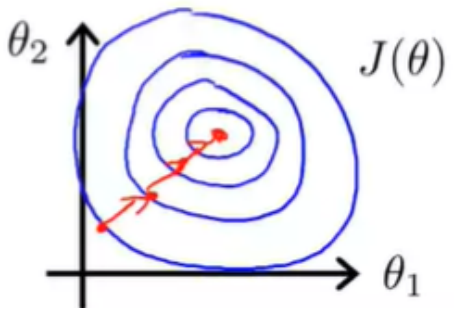
x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$

$0 \leq x_1 \leq 1$ $0 \leq x_2 \leq 1$



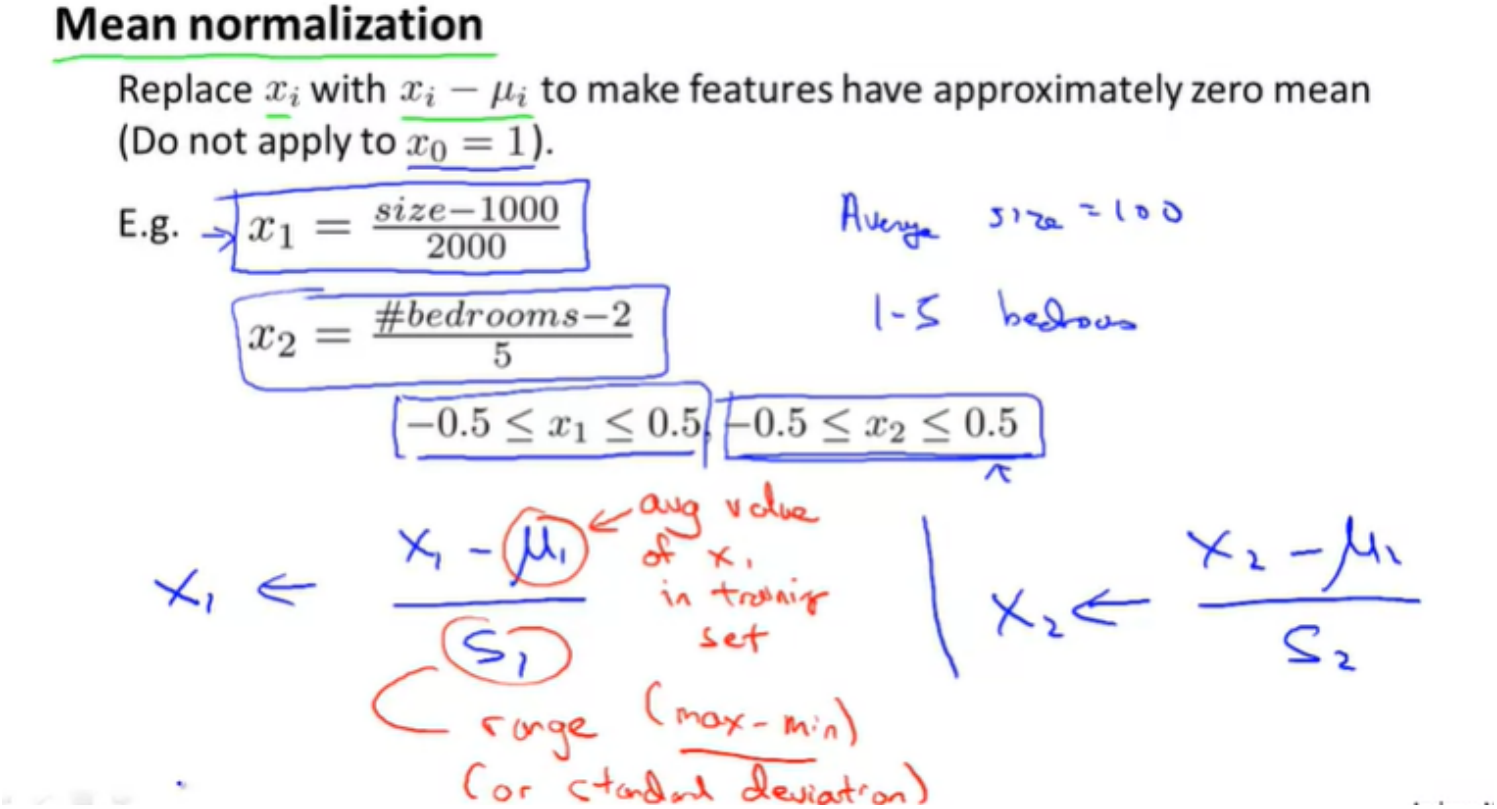
on the left image, the global minima takes more time to be found. due to narrow contour lines and variable gaps

where as the image on the right will be give better solutiona as it

equally arranged for the minima

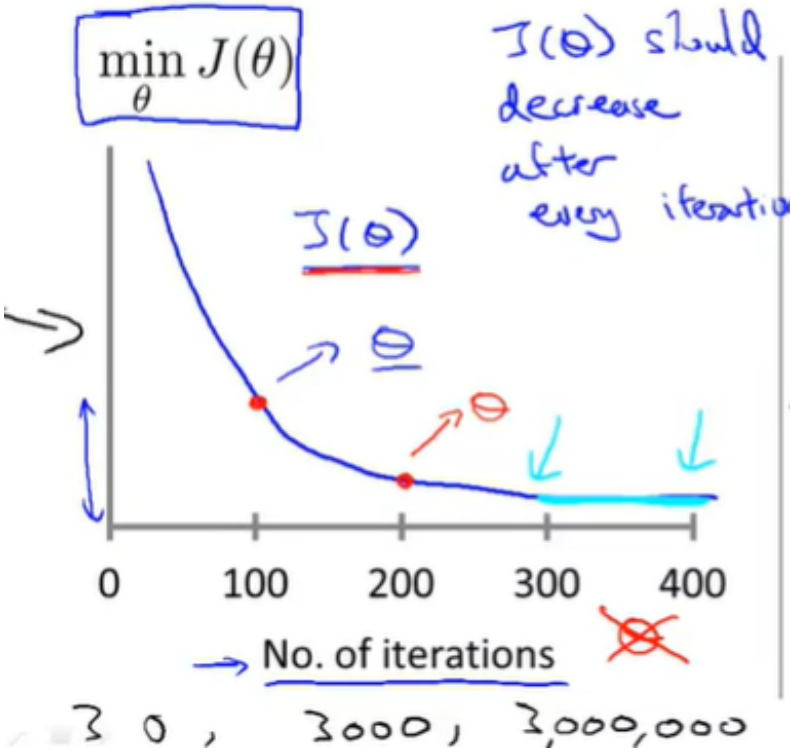
we should get every feature into approx. a -1 <= x(i) <= 1 range
make sure the range is [-1,1] and have reasonale decimal value

Another way is to use :
Mean Normalization
Replace the x(i) with x(i) - U(i) to make features have approx zero mean (do not apply x(0) =1)



Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero

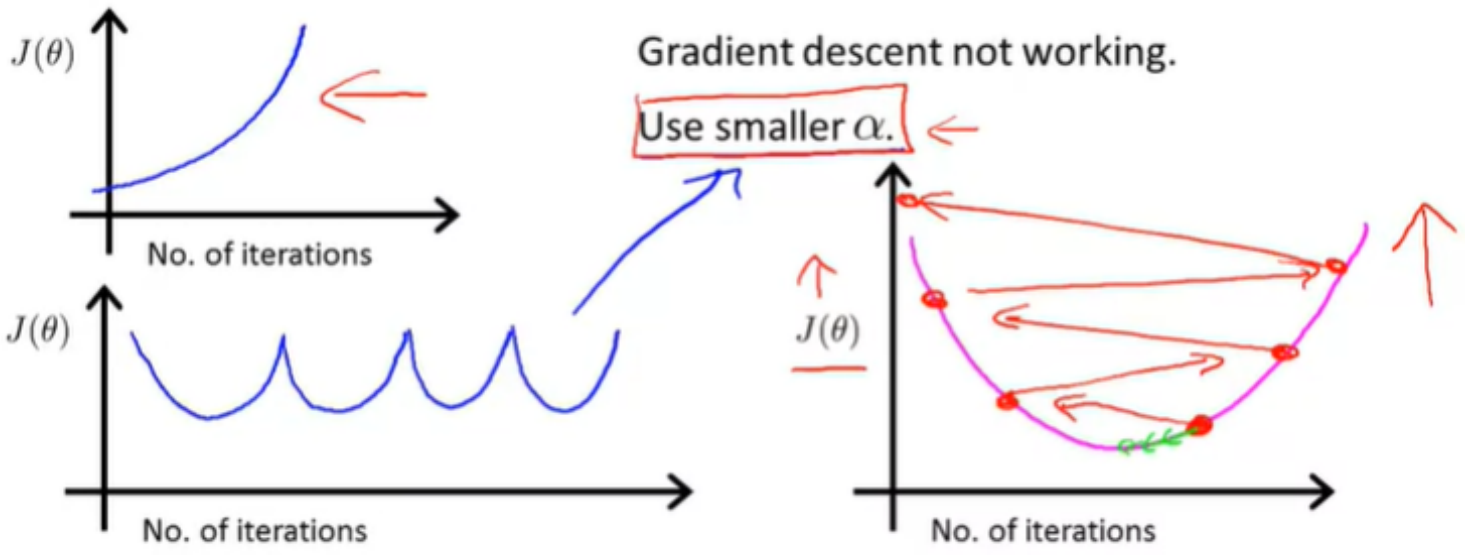
Learning rate : gradient descent in practice II
debugging to make sure the gradient descent is working i.e choose learning rate



→ Example automatic convergence test:

→ Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

ε



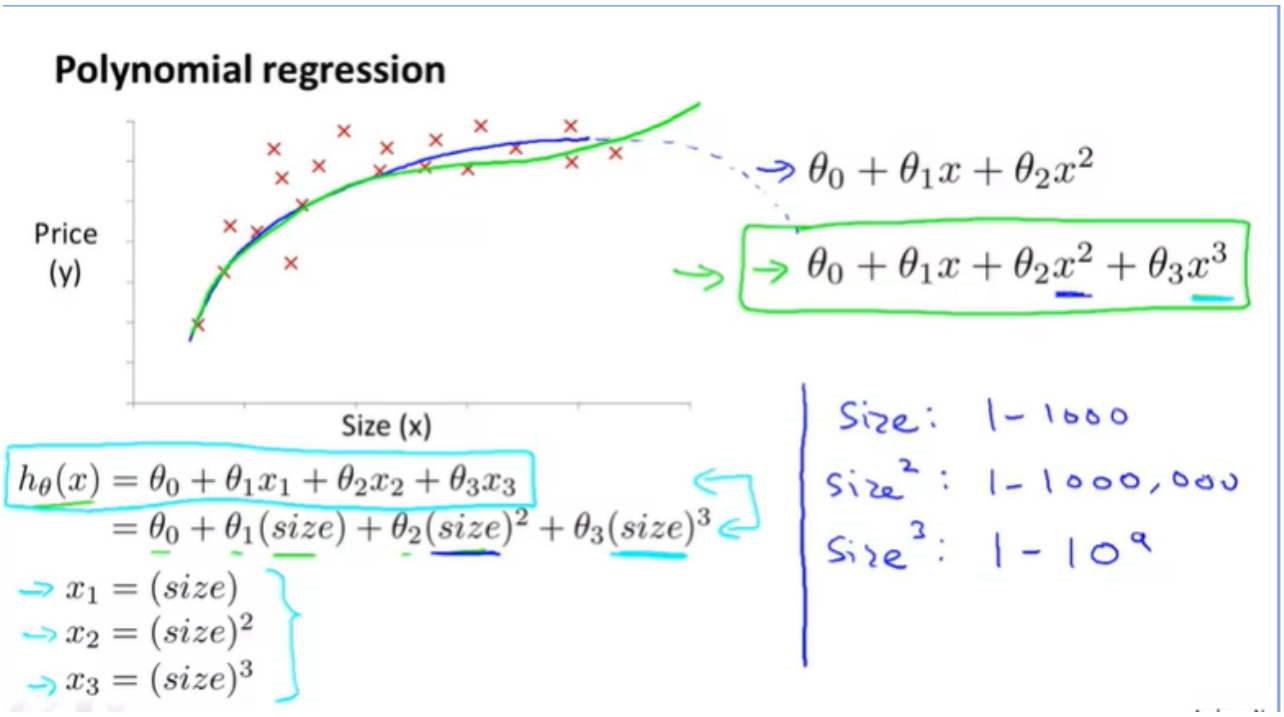
- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

if a is too small : slow convergence
...0.001... 0.01...0.1....1...

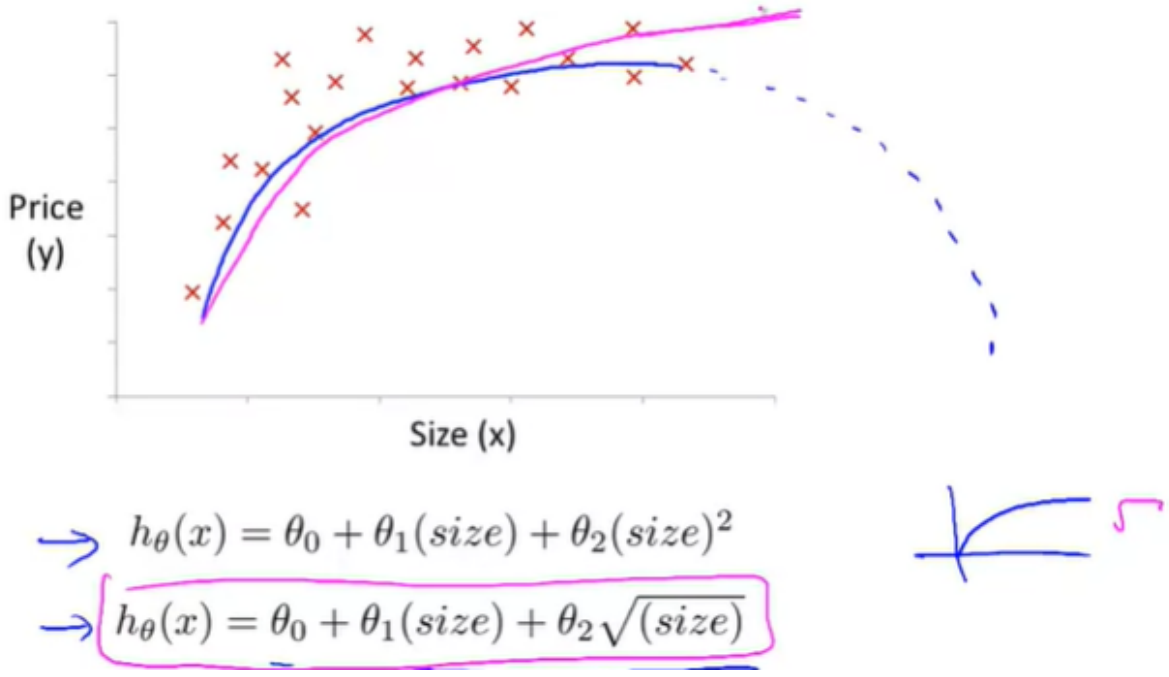
Features and Polynomial regression

suppose two features are length and breadth

we can consider one feature Area for the above ; understand the need of and relation of features for optimization



problem : A quadratic function is increasing but may decrease later
So, implement different choice of features or implement change in function
choice of feature



Computing Parameters analytically
Normal Equation
method to solve for Theta analytically - i.e in one step instead of iterative
In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj ’s, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:
θ= (X^T * X)^(-1) * (X^T) *y

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

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

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

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

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

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

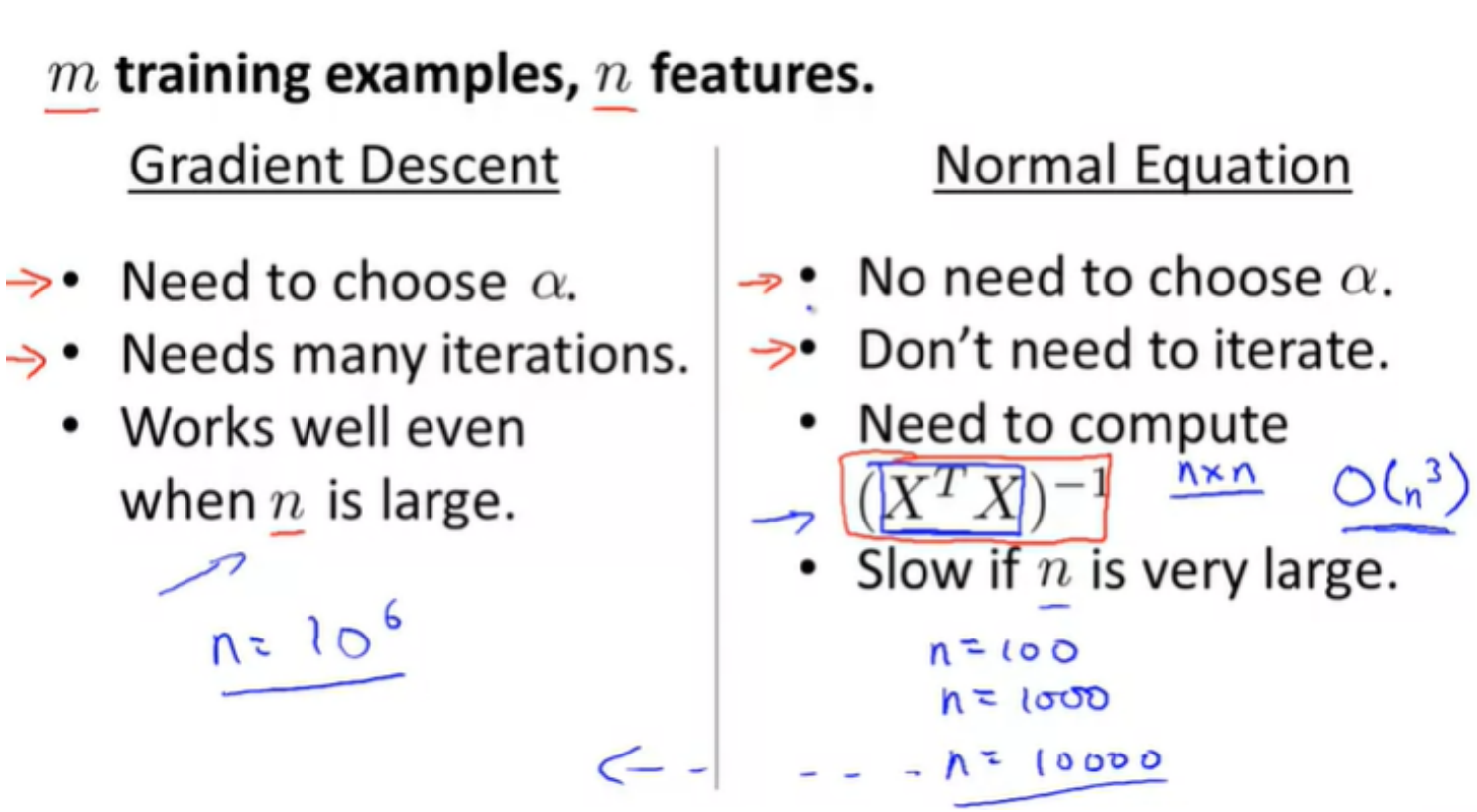
X (design matrix) $= \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$ $m \times (n+1)$

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix} \quad m \times 2 \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

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

no need of feature scaling with normal equation



Normal Equation Invertibity

if the matrix is invertible (X^T * X)

this is caused due to :

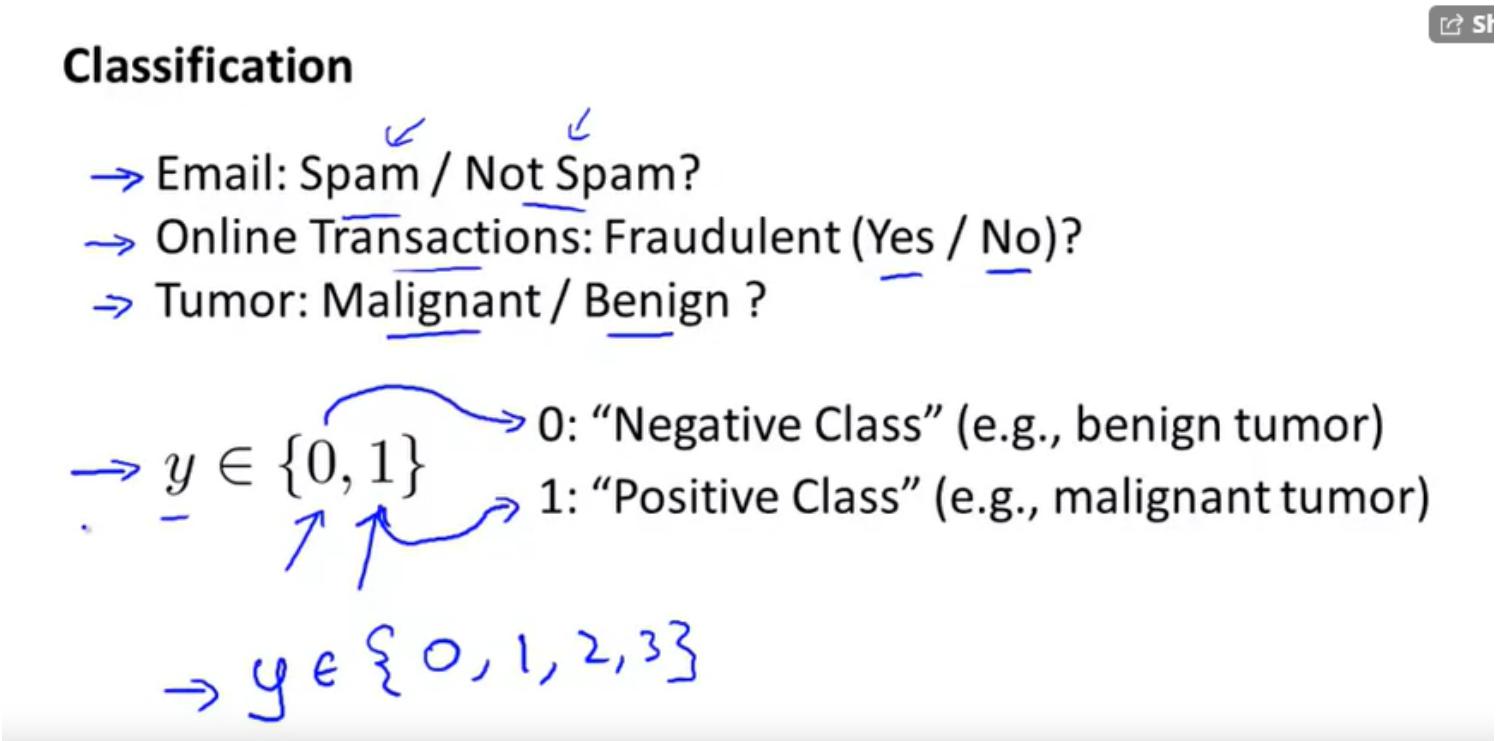
1. reduntant features (linearly dependent)
   two features are very closely related
2. too many features (m>=n)
   delete some features or use regularization

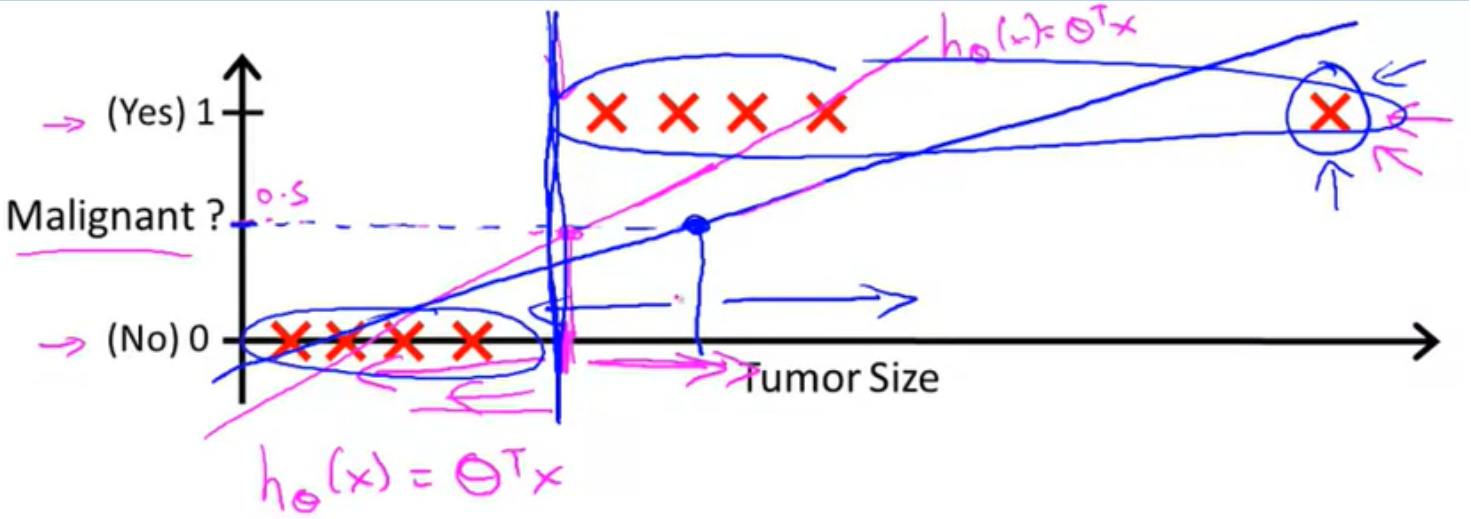
When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv.' The 'pinv' function will give you a value of \theta$\theta$ even if X^TXX^TX is not invertible

Week3

Logistic Regression
Classification problems
    EMAIL: spam / not spam
    Online Transactions : Fradulent or not
    tumor malignant / benign

binary classification problem 0,1
multiple classification problem 0,1,2,3,4..





→ 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"

not a good idea to use linear regression for classification problem

Classification: y = 0 or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

(Classification)

Hypothesis Representation

logistic Regression model
sigmoud/logistic function

asymtotes at 0,1

fit the parameters theta to make predictions

Logistic Regression Model

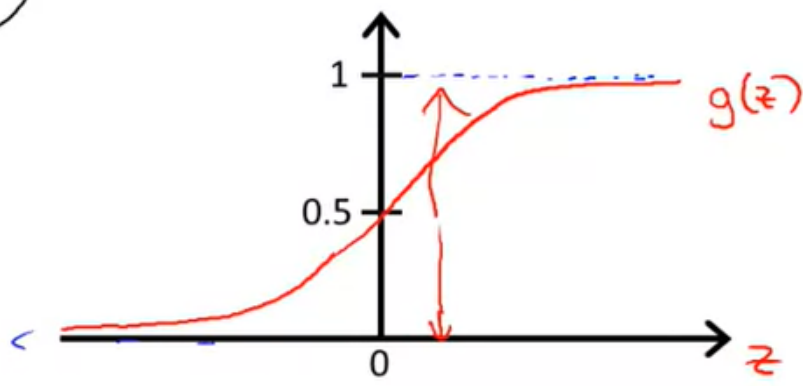
Want $0 \leq h_\theta(x) \leq 1$

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

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

$z = \theta^T x$

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



Sigmoid function

Logistic function

Parameters θ

Intrepretation of hypothesis output

Interpretation of Hypothesis Output

$h_\theta(x)$

$h_\theta(x)$ = estimated probability that y = 1 on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$

$y = 1$

Tell patient that 70% chance of tumor being malignant

$h_\theta(x) = P(y=1|x;\theta)$

"probability that y = 1, given x, parameterized by θ"

$y = 0$ or 1

$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$

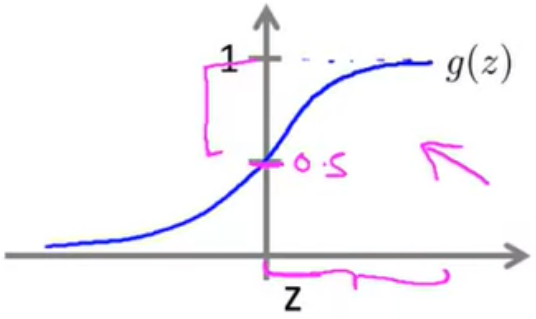
$P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$

Decision Boundary

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 $h_\theta(x) \geq 0.5$

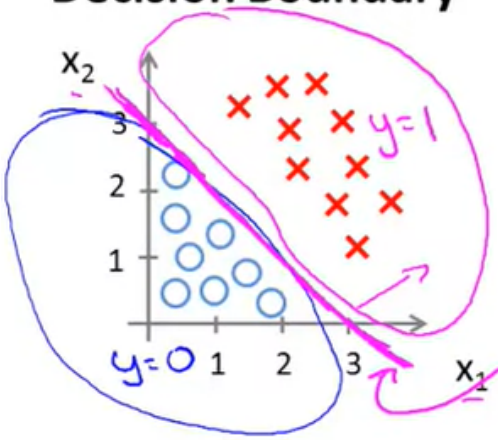
$g(z) \geq 0.5$ when $z \geq 0$

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

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

Decision Boundary

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



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

$-3 \quad 1 \quad 1$

Decision boundary

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

$\theta^T x$

$x_1 + x_2 \geq 3$

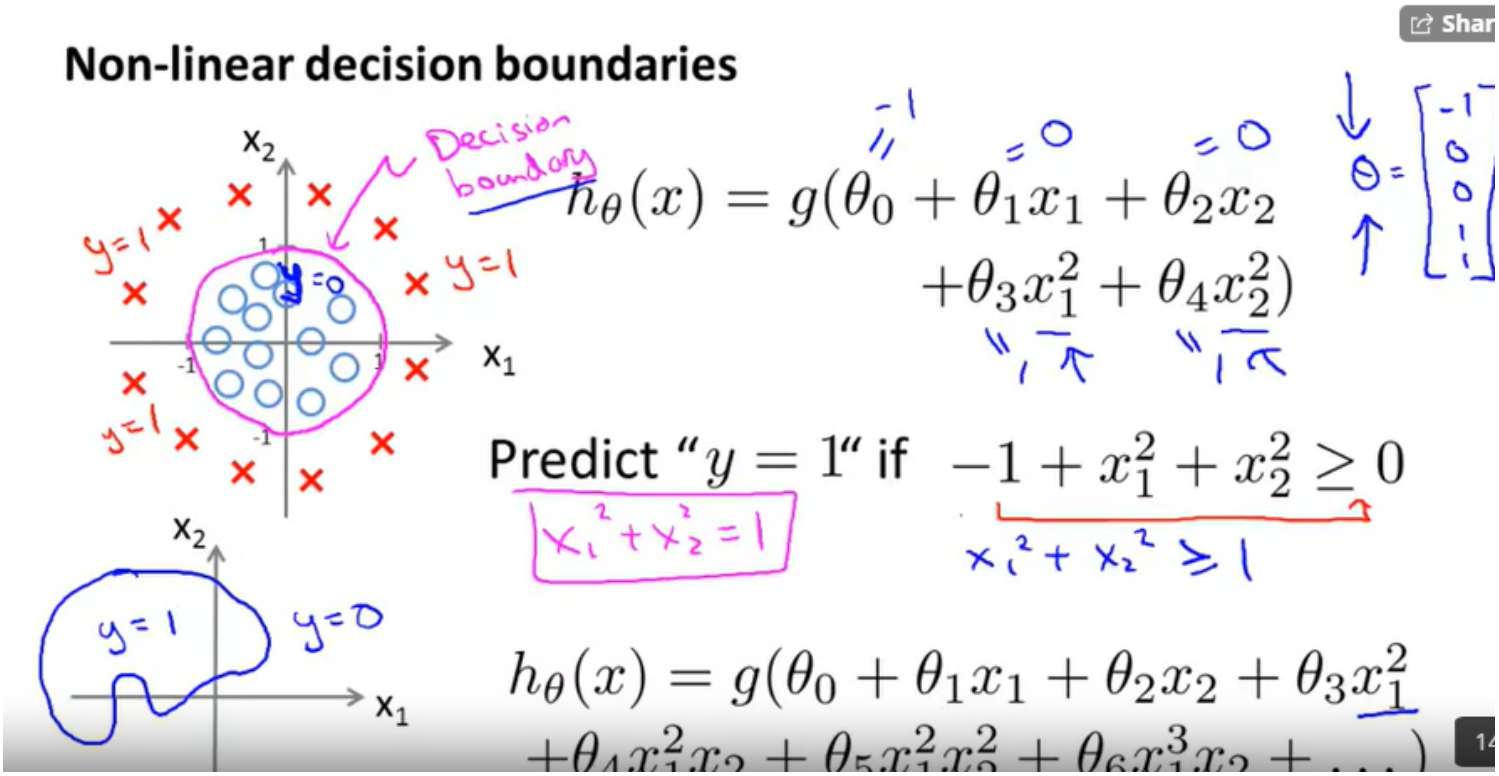
x_1, x_2

$h_\theta(x) = 0.5$

$x_1 + x_2 = 3$

$x_1 + x_2 < 3$

$y = 0$



The **decision boundary** is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

Example:

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$

$$y = 1 \; if \; 5 + (-1)x_1 + 0x_2 \geq 0$$
$$5 - x_1 \geq 0$$
$$-x_1 \geq -5$$
$$x_1 \leq 5$$

In this case, our decision boundary is a straight vertical line placed on the graph where x_1 = 5$x1$=5, and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.
Again, the input to the sigmoid function g(z) (e.g. \theta^T XθTX) doesn't need to be linear, and could be a function that describes a circle (e.g. z = \theta_0 + \theta_1 x_1^2 +\theta_2 x_2^2z=θ0+θ1 x12+θ2x22) or any shape to fit our data.

Logistic regression Model

Cost Function

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 ?

non-complex - non linear function from the sigmod substitution to h(x) makes it give many local minima

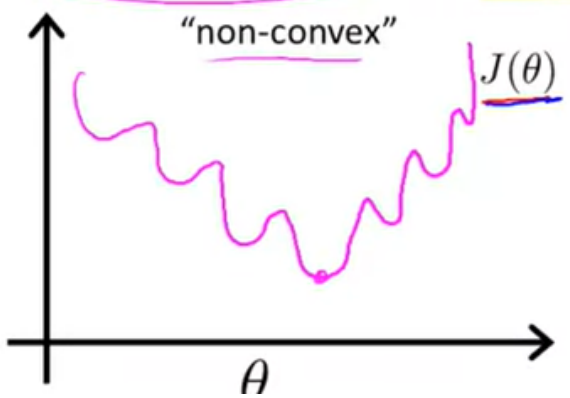
Cost function

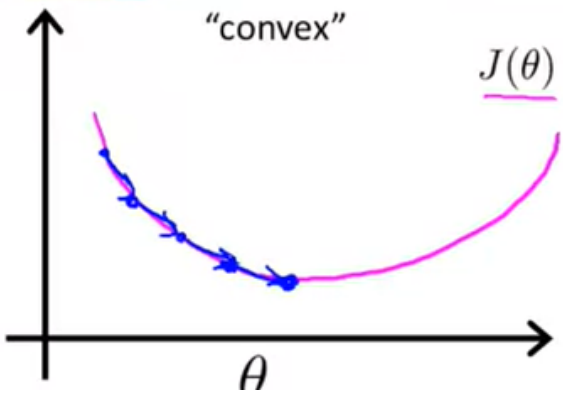
-> ~~Linear~~ regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Logistic

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

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

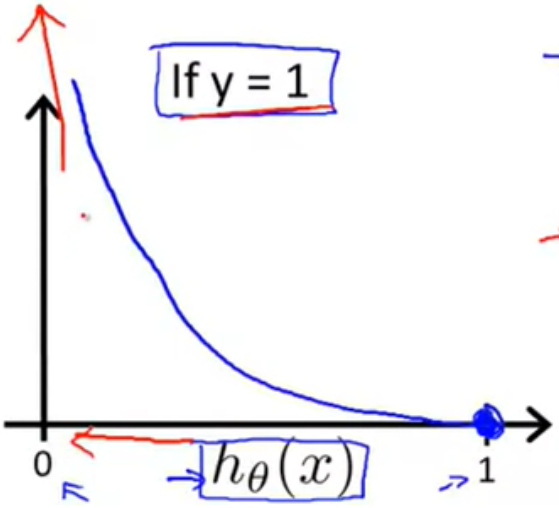




logistic regression cost function

Logistic regression cost function

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



- $\text{Cost} = 0$ 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.

convexity analysis is part of the topics- convex optimization problem

If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.

If our correct answer 'y' is 1, then the cost function will be 0 if our hypothesis function outputs 1. If our hypothesis approaches 0, then the cost function will approach infinity.

Note that writing the cost function in this way guarantees that J(θ) is convex for logistic regression.

Simplified Cost function and Gradient descent

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

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

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

If $y = 0$: $\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x))$

derived from statistics from the principle of maximum likelihood estimation

simplifying the cost function

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

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

To fit parameters θ:

$$\min_\theta J(\theta)$$ Get θ

To make a prediction given new x:

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

we use gradient descent to minimize the cost function

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

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

Repeat {

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

(simultaneously update all θ_j)

}

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

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

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

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$ for $i = 0$ to n

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

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

Algorithm looks identical to linear regression!

feature scaling another approach

Repeat $\{$

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

$\}$

We can work out the derivative part using c

Repeat $\{$

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

$\}$

Notice that this algorithm is identical to the theta.

A vectorized implementation is:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

Advanced Optimization

- advance numerical computing

finding a good library for any of these implementation of the advance or normal functions usage to yeild better and faster result

Optimization algorithm

Given θ, we have code that can compute

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

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

fminunc - function minimization unconstrainted

Example:

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

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

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

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

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

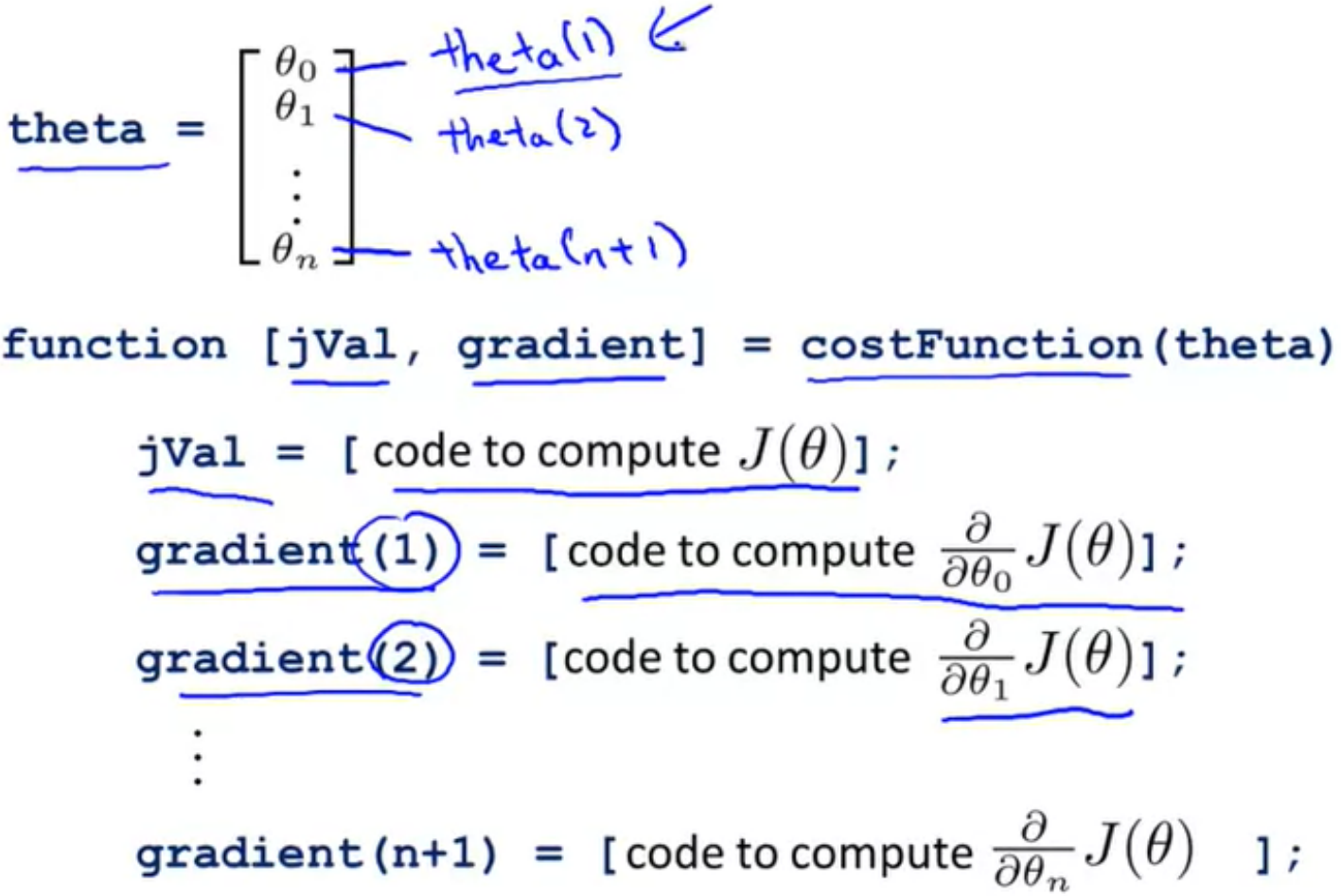
```
function [jVal, gradient]
            = costFunction(theta)
  jVal = (theta(1)-5)^2 + ...
         (theta(2)-5)^2;
  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

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

options = optimset('GradObj','on','MaxIter','100');

```
octave-3.2.4.exe:1> PS1('>> ')
>> cd 'C:\Users\ang\Desktop'
>>
>> options = optimset('GradObj','on', 'MaxIter', '100');
>> initialTheta = zeros(2,1)
initialTheta =

   0
   0

>> [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options)
optTheta =

   5.0000
   5.0000

functionVal = 1.5777e-030
exitFlag =  1
```

cost function for logistic regression from gradient descent



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

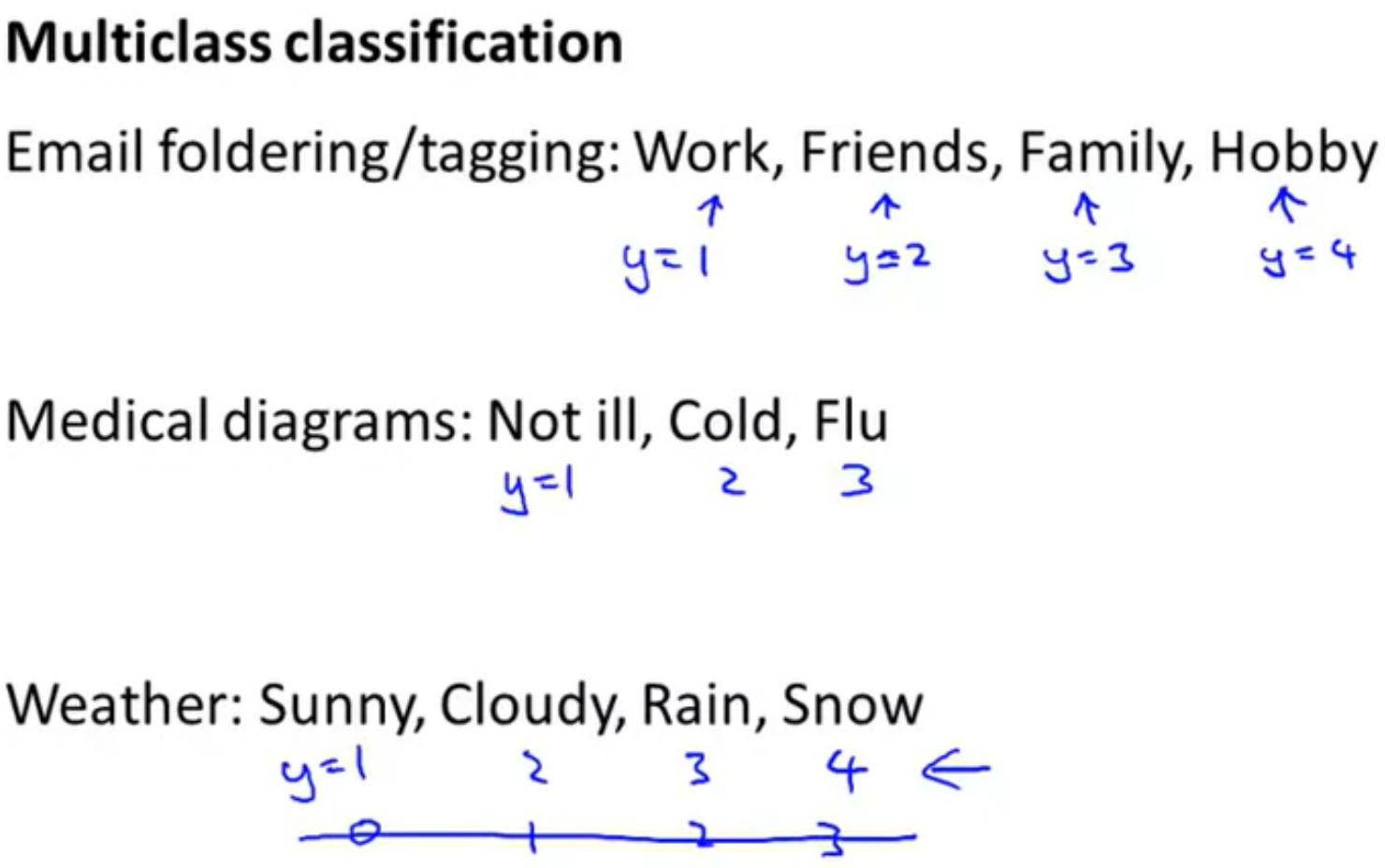
```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

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)

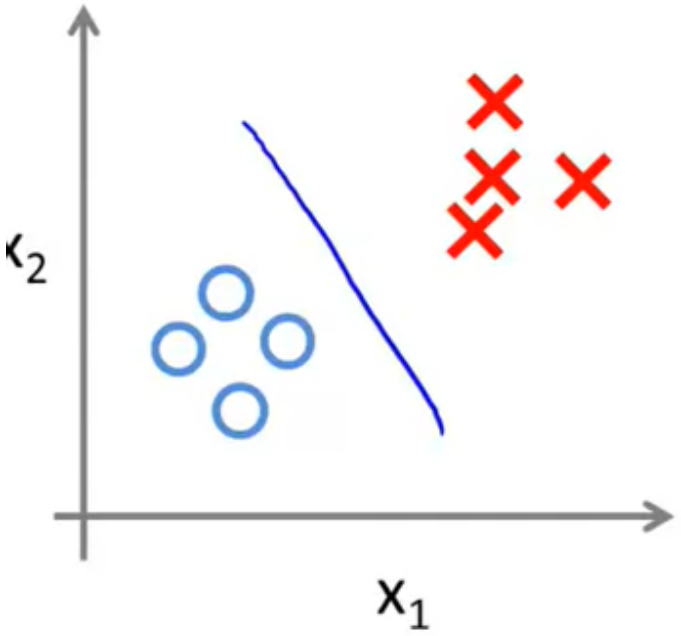
```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);

```

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

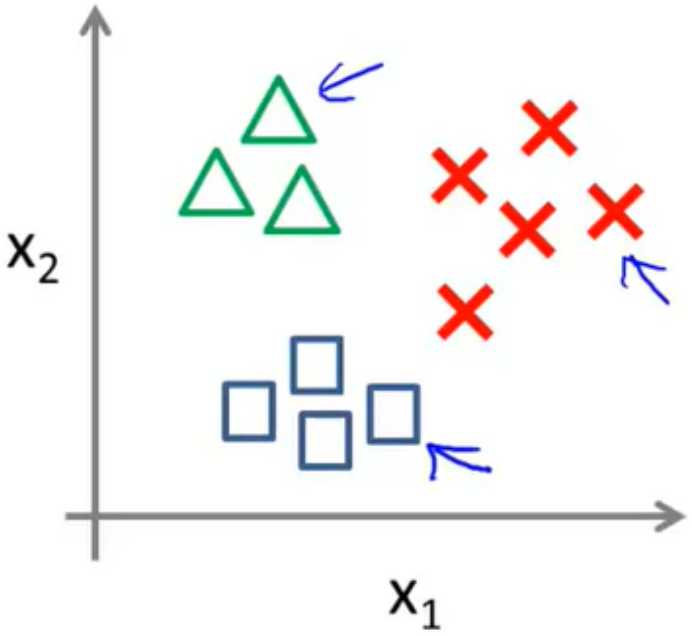
Multiclass Classificiation - one vs All



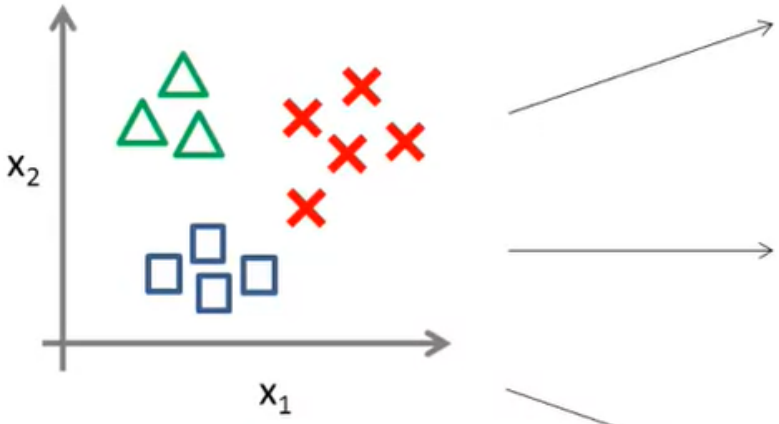
Binary classification:



Multi-class classification:

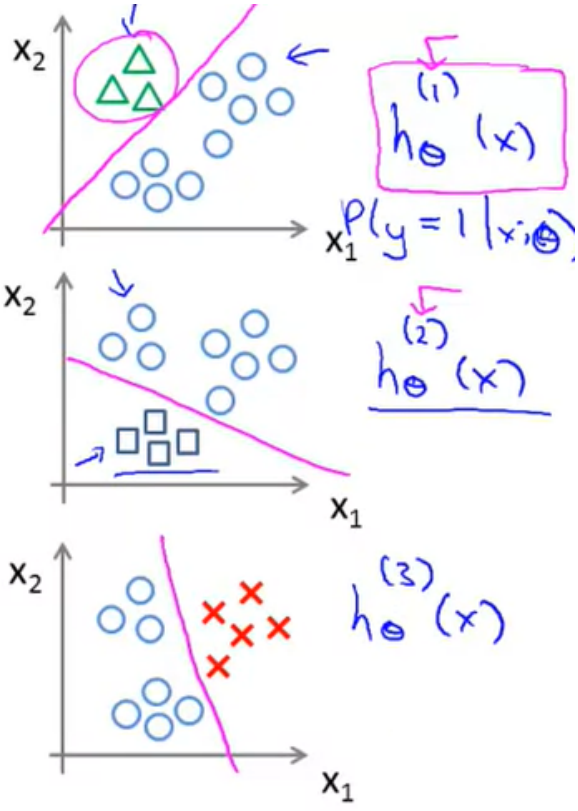


One-vs-all (one-vs-rest):



Class 1: △ ←
Class 2: □ ←
Class 3: ✕ ←

$h_\theta^{(i)}(x) = P(y = i|x;\theta) \quad (i = 1, 2, 3)$



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

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

Whatever value gives the highest probability, we consider that class

Now we will approach the classification of data when we have more than two categories. Instead of y = {0,1} we will expand our definition so that y = {0,1...n}.

Since y = {0,1...n}, we divide our problem into n+1 (+1 because the index starts at 0) binary classification problems; in each one, we predict the probability that 'y' is a member of one of our classes.

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

We are basically choosing one class and then lumping all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction.

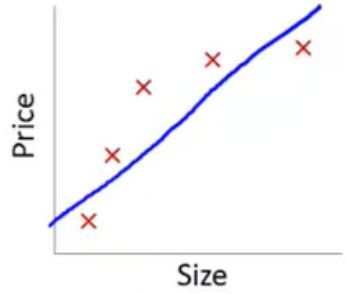
To summarize:

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

To make a prediction on a new x, pick the class that maximizes $h_\theta(x)$

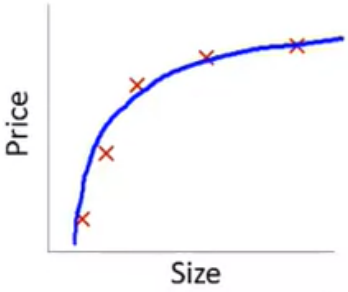
Problem of Overfitting

Example: Linear regression (housing prices)



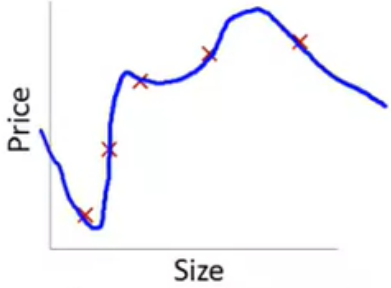
$\rightarrow \theta_0 + \theta_1 x$

"Underfit" "High bias"



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

"Just right"



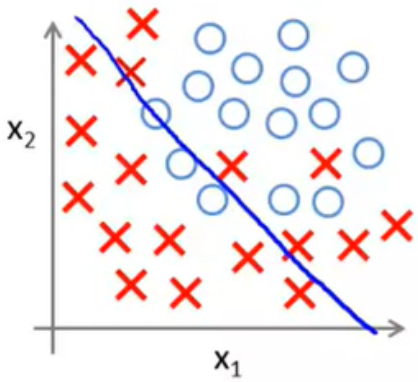
$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

"Overfit" "High variance"

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).

fit a linear function - not a great model - underfitting - high bias
quadratic function - better model - just right
polynomial function - overfitting - high variance
- fails to generalize the new examples

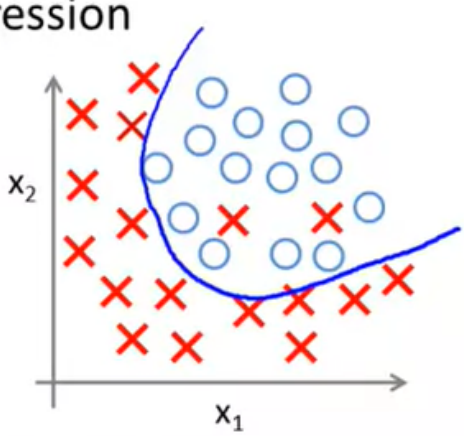
Example: Logistic regression



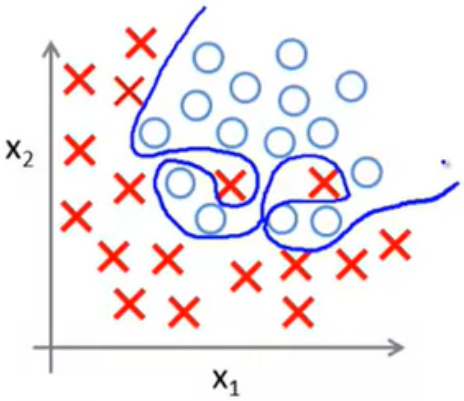
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

"Underfit"



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

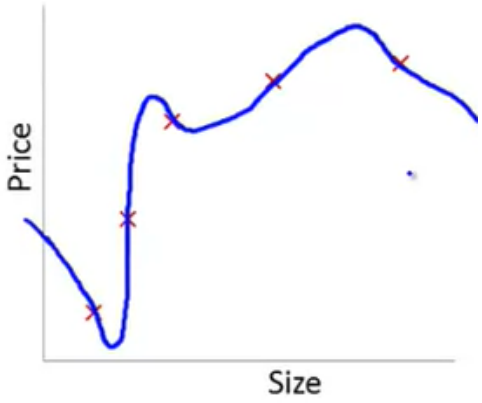


$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

addressing overfitting
- problems due to alot of features and cause problems in plotting and visualization
- smaller training set and alot of faetures - causes overfitting

Addressing overfitting:

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}



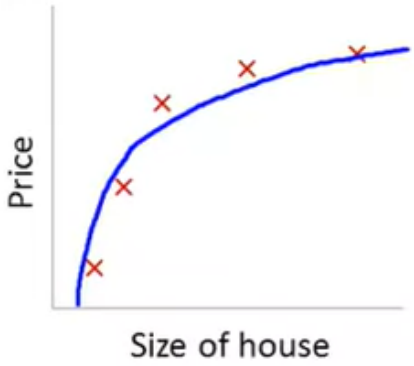
options to overcome the problem :

Options:

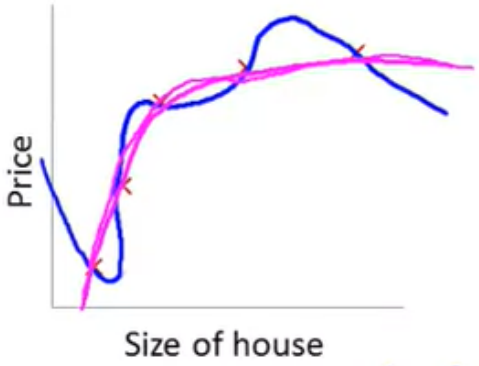
1. Reduce number of features.
   - — Manually select which features to keep.
   - — Model selection algorithm (later in course).
2. Regularization.
   - — Keep all the features, but reduce magnitude/values of parameters θ_j.
   - — Works well when we have a lot of features, each of which contributes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

Regularization - Cost Function

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.

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

$\theta_3 \approx 0 \qquad \theta_4 \approx 0$

minimizing the function with more parameter by nulling them to zeros

shrink all the parameters

Regularization.

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

- "Simpler" hypothesis
- Less prone to overfitting

$\theta_3, \theta_4 \approx 0$

Housing:

- Features: $x_1, x_2, \ldots, x_{100}$
- Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}$

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

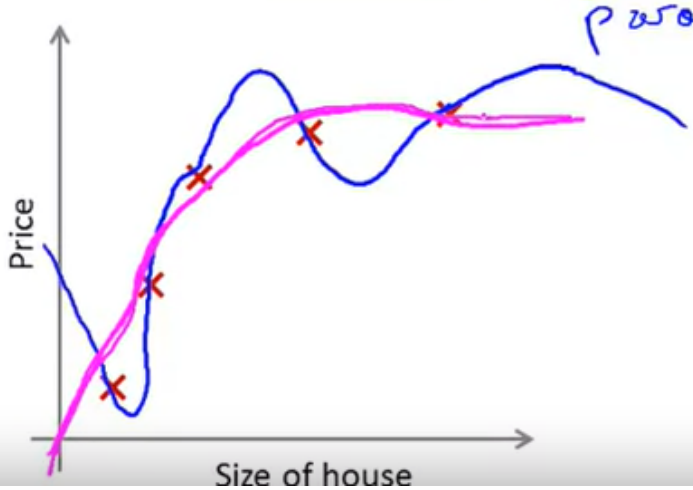
$\theta_1, \theta_2, \theta_3, \ldots, \theta_{100}$

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

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



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 $\lambda = 10^{10}$)?

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

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

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

$h_\theta(x) = \theta_0$

$h_\theta(x)$

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

Regularized Linear Regression

Gradient descent

Repeat {

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

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

$(j = 1, 2, 3, \ldots, n)$

}

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

$1 - \alpha \frac{\lambda}{m} < 1$ — 0.99 — $\theta_j \times 0.99$

Normal equation

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

$m \times (n+1)$

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

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

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

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

$(n+1) \times (n+1)$

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

Non-invertibility (optional/advanced).

Suppose $m \leq n$,

(#examples) (#features)

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

non-invertible / singular

pinv

inv

If $\lambda > 0$,

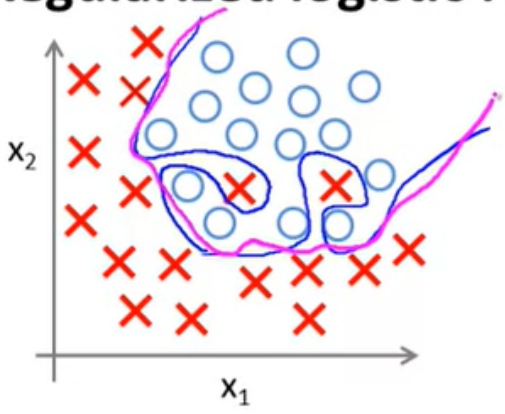
$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

this matrix will not be singular (...)

Recall that if m < n, then X^TX*XTX* is non-invertible. However, when we add the term λ·L, then X^TX*XTX* + λ·L becomes invertible.

Regularized Logistic Regression

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:

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

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

Gradient descent

Repeat {

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

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

$(j = 1, 2, 3, \ldots, n)$

$\theta_1 \ldots \theta_n$

}

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

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

advanced optimization methods

Advanced optimization

fminunc (@costFunction, θ = [θ_0 theta(1); θ_1 theta(2); θ_n theta(n+1)])

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(theta)];
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
    gradient(3) = [code to compute ∂/∂θ2 J(θ)];
    :
    gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

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

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

$\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} \right) + \frac{\lambda}{m} \theta_1$

$\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)} \right) + \frac{\lambda}{m} \theta_2$

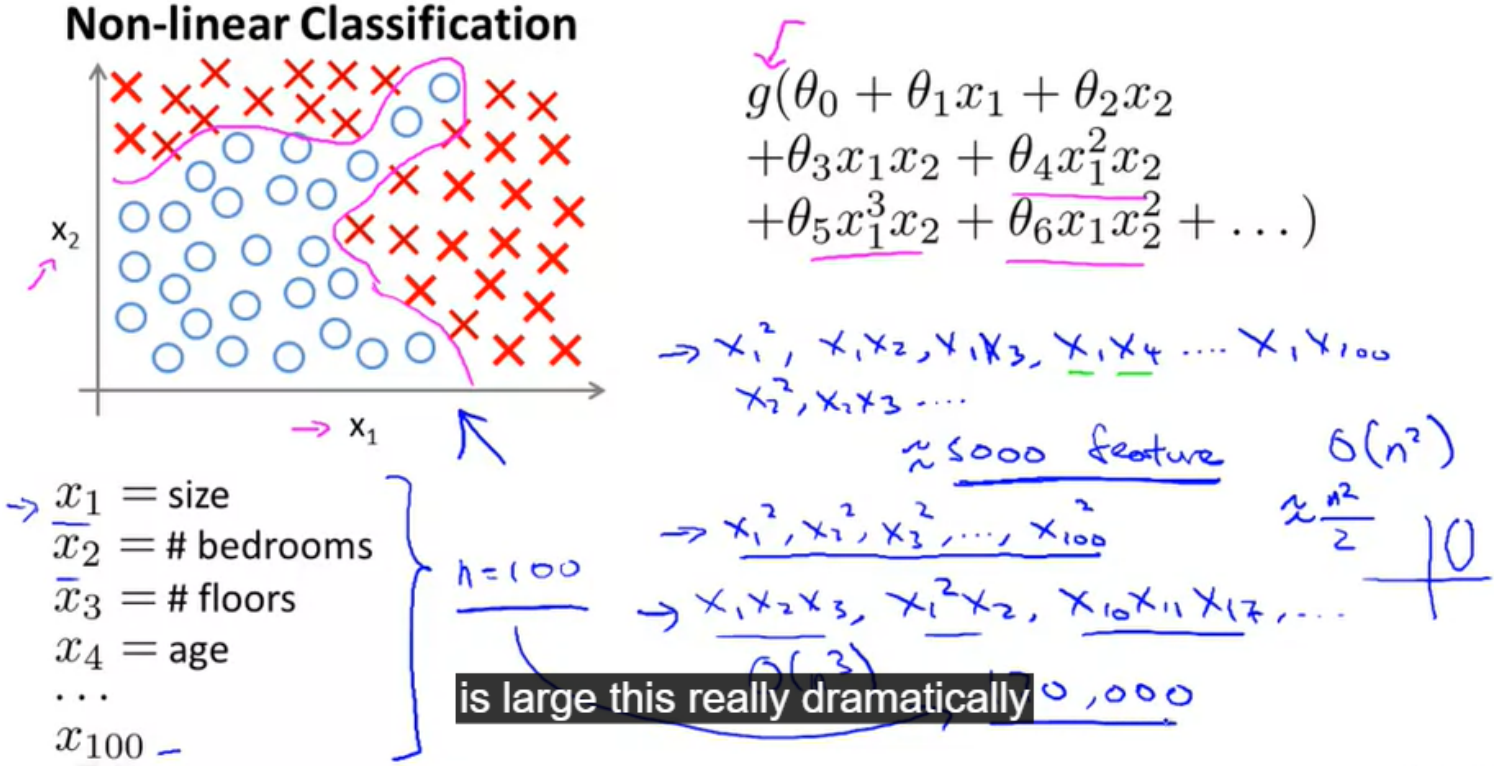
$J(\theta)$

Week4

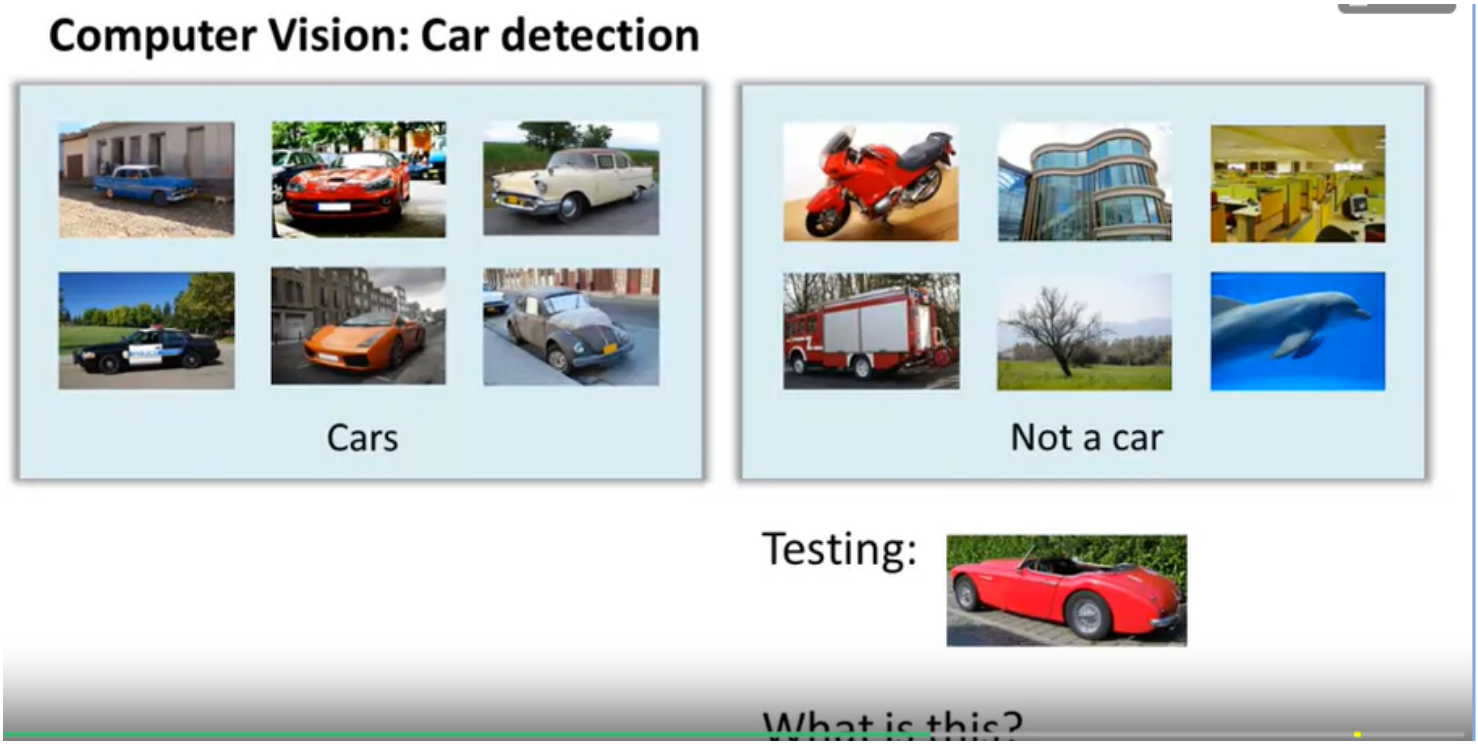
Neural Networks

Nonlinear Hypotheses

Learning algorithm
Supervised learning - logistic regression

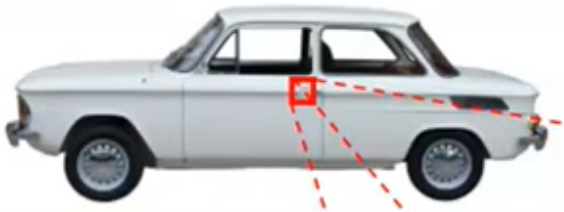


non-linear classification



What is this?

You see this:

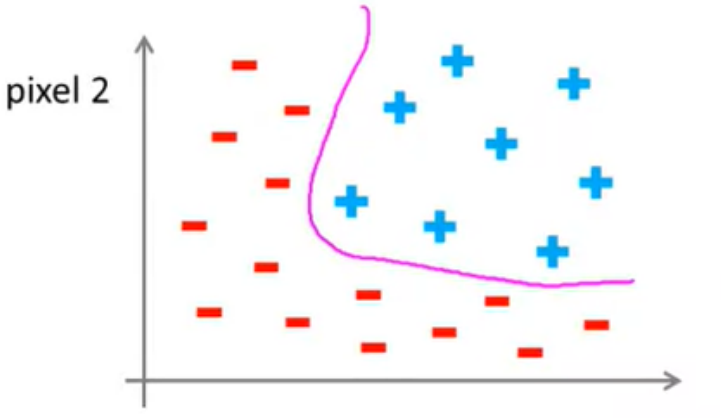


But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
								59	75	69	50

pixel intensity values, and tell

Learning Algorithm



50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

0-255

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

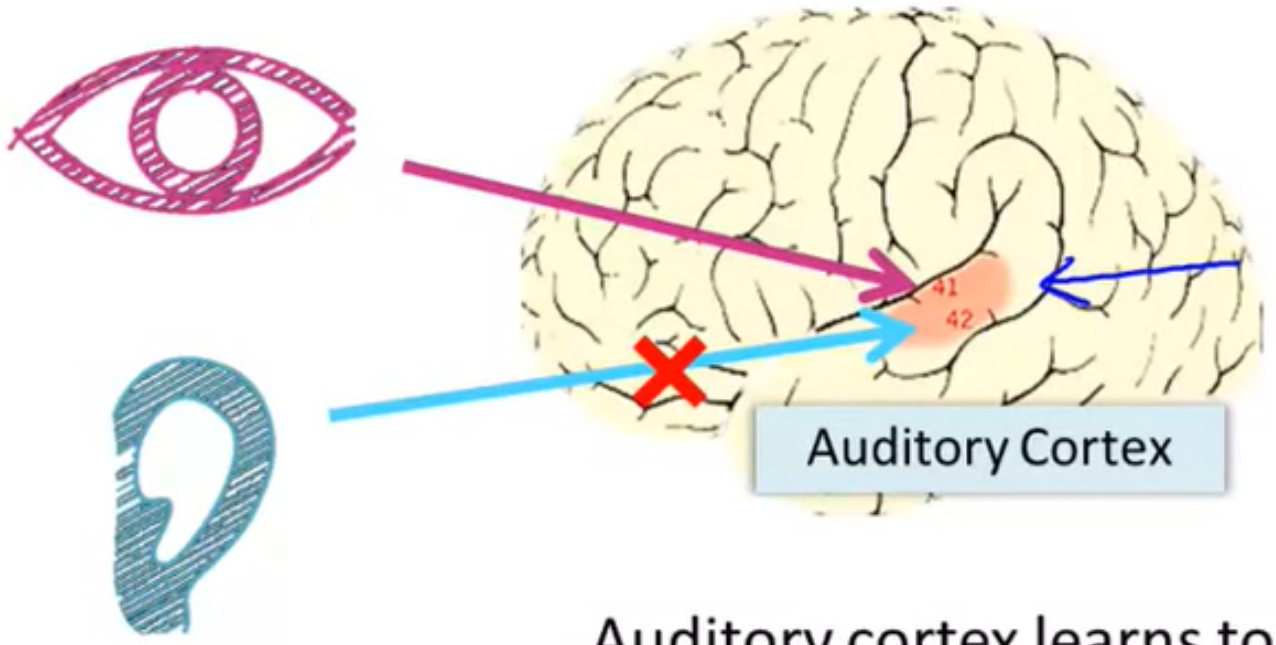
Quadratic features ($x_i \times x_j$): ≈3 million features

it is too large to solve fast

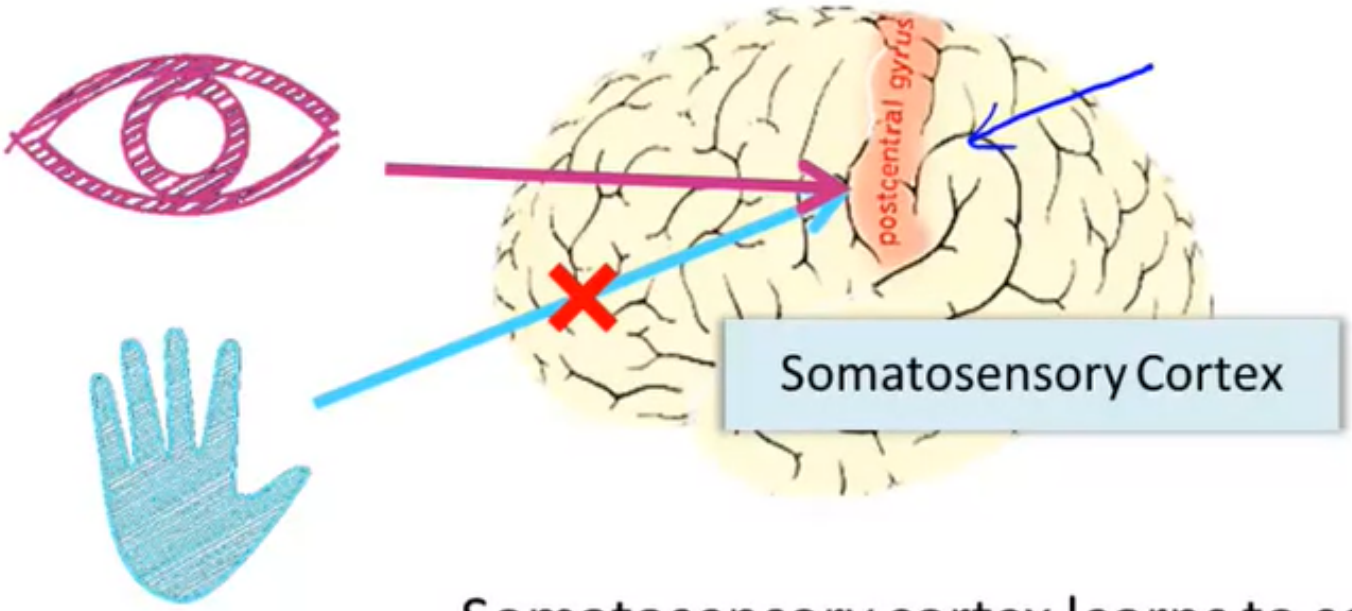
Neurons and brain
computationally expensive algorithms

Neural Networks

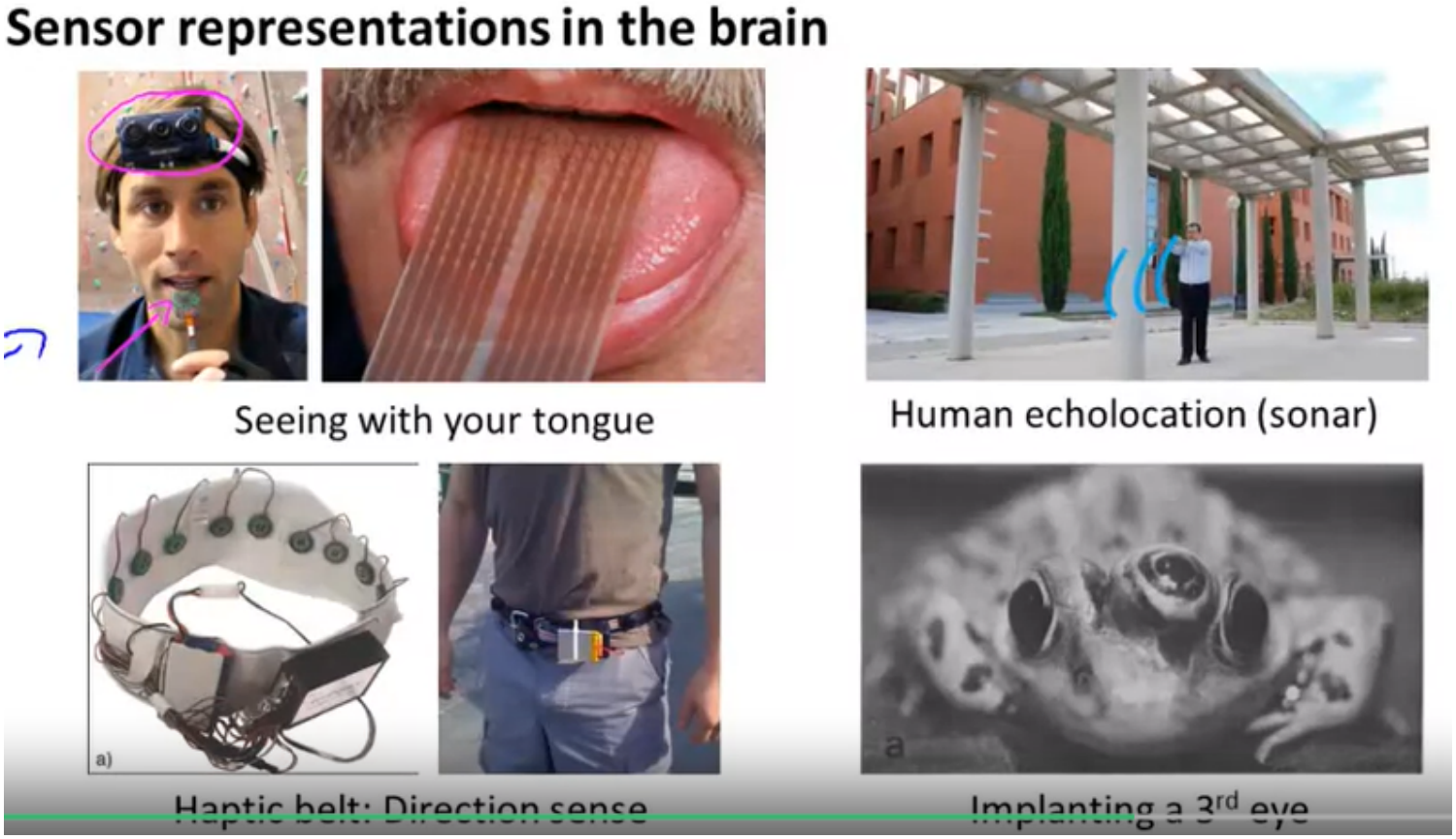
- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications



Auditory cortex learns to see



Somatosensory cortex learns to see

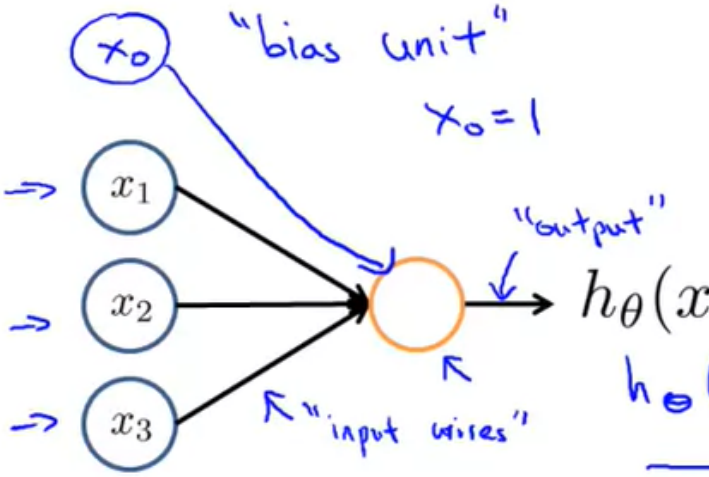


Neural Networks
Model representation I

represent hypotheses or model

neuron model : logistic unit
X0 is bias unit. outputs the value 1
weights: paramenters

Neuron model: Logistic unit



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

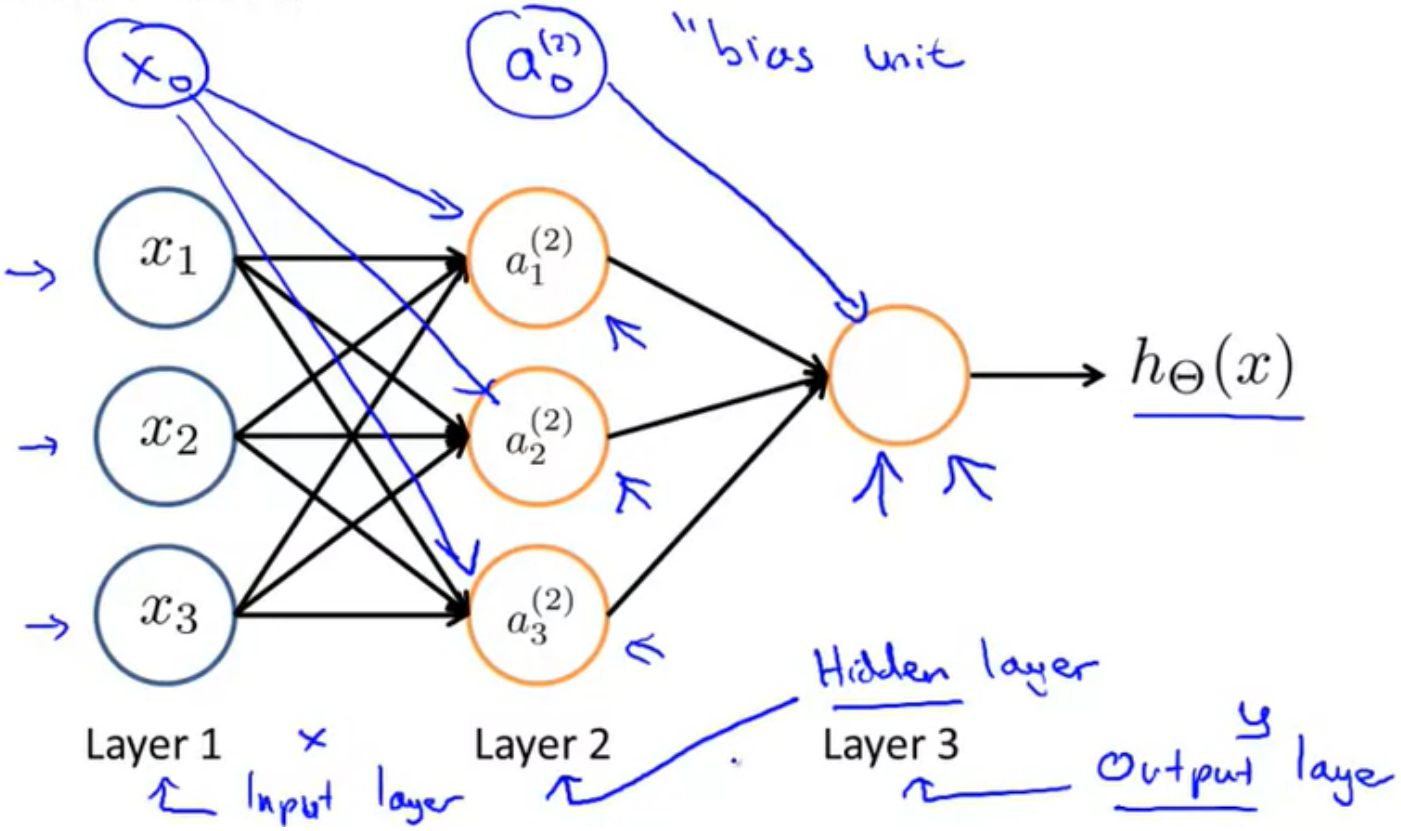
"weights" (parameters)

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

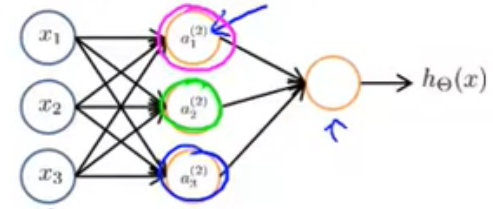
Sigmoid (logistic) activation function.

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

Neural Network



Neural Network



3 units 3 hidden units

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

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

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

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

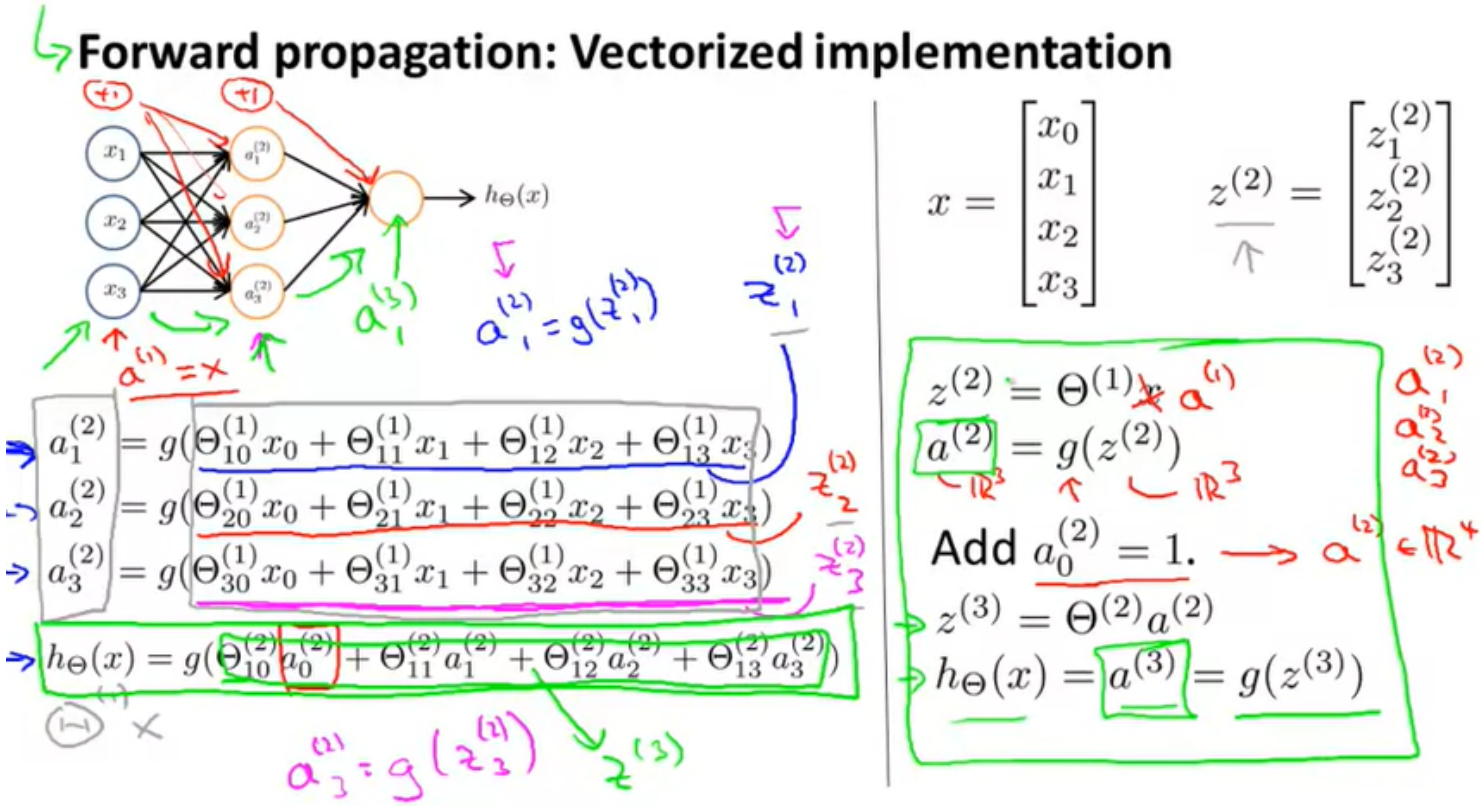
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

→ If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

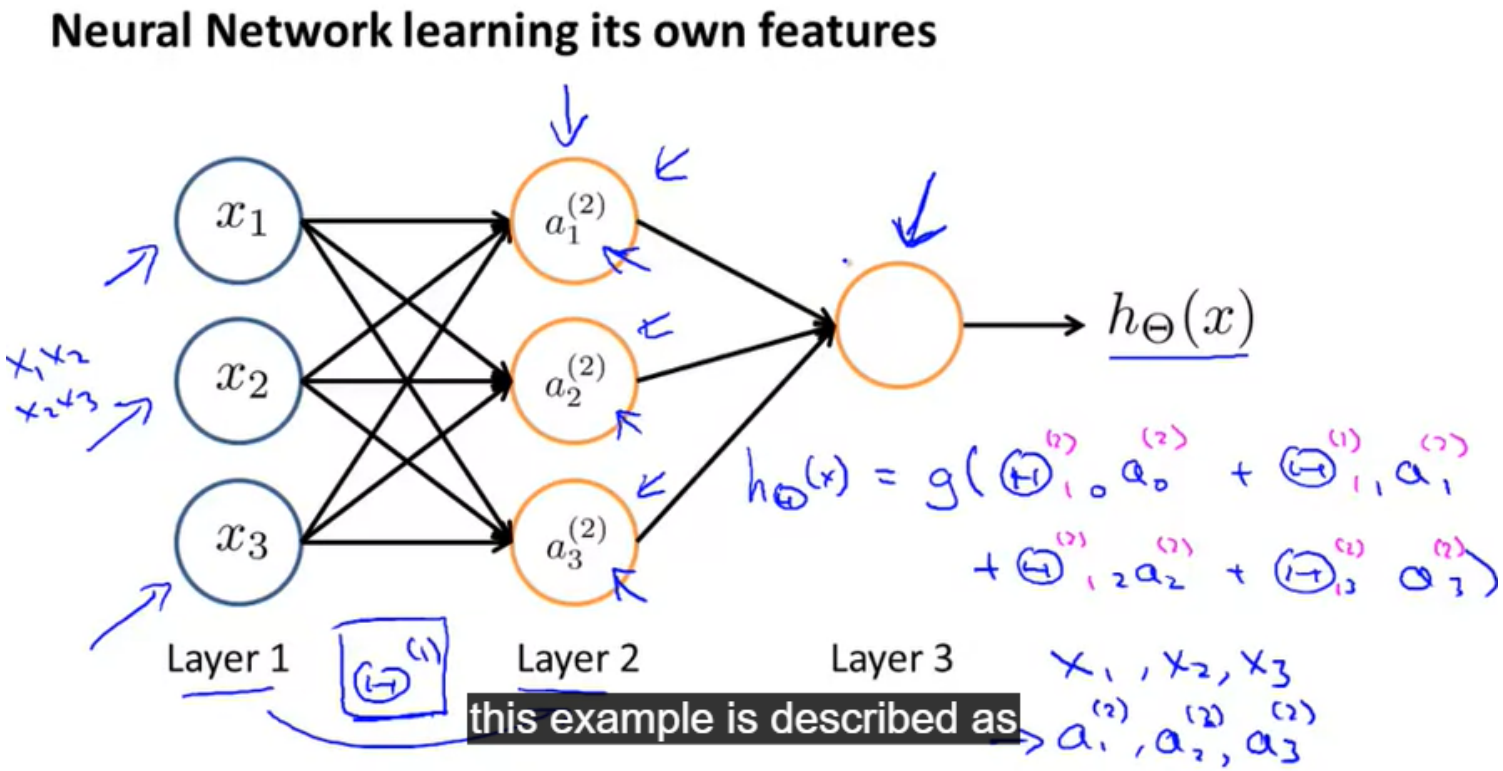
At a very simple level, neurons are basically computational units that take inputs (dendrites) as electrical inputs (called "spikes") that are channeled to outputs (axons). In our model, our dendrites are like the input features x_1\cdots x_nx1⋯xn-, and the output is the result of our hypothesis function. In this model our x_0x0 input node is sometimes called the "bias unit." It is always equal to 1. In neural networks, we use the same logistic function as in classification, \frac{1}{1 + e^{-\theta^Tx}}1+e−θTx1, yet we sometimes call it a sigmoid (logistic) activation function. In this situation, our "theta" parameters are sometimes called "weights".

Model representation II

forward propagation : vectorized implementation
linear combination that go into a particular neuron

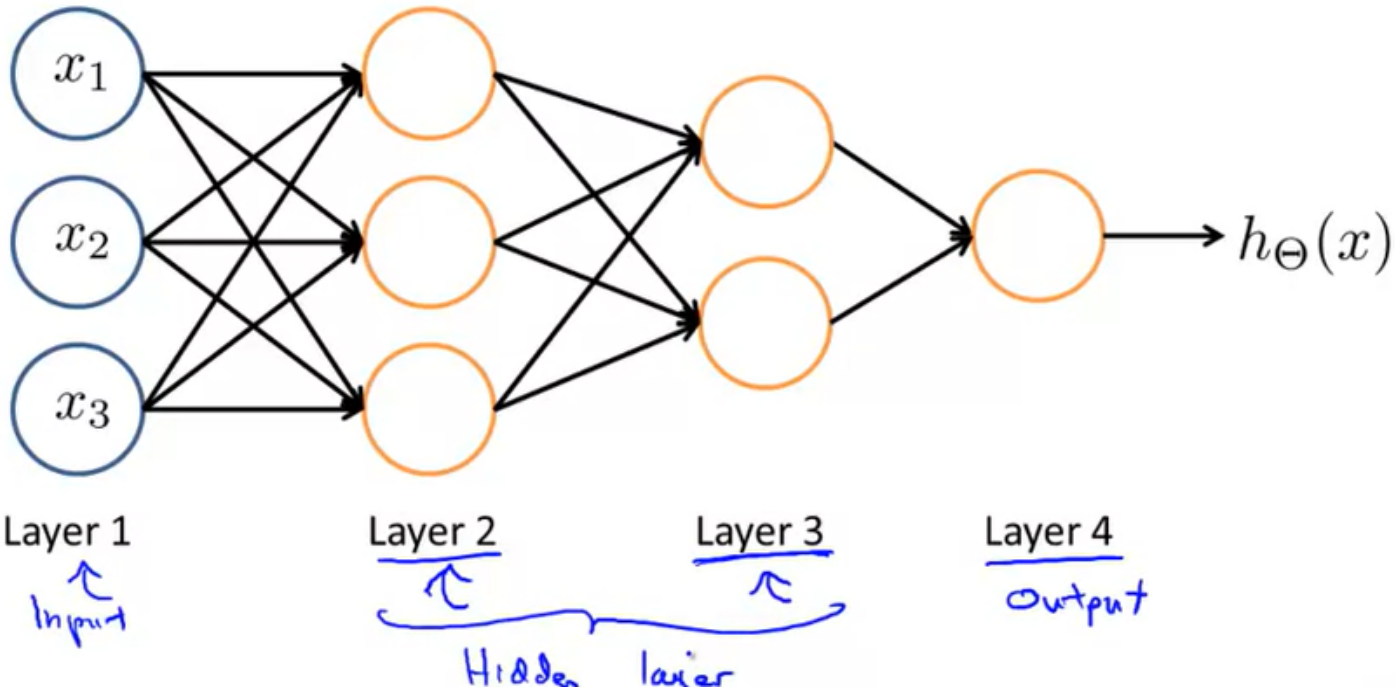


neural networks learning its own features



other network architectures

Other network architectures



In this section we'll do a vectorized implementation of the above functions. We're going to define a new variable z_k^{(j)}*zk(j)* that encompasses the parameters inside our g function.

$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
\end{aligned}
$$

$$
\begin{aligned}
a_1^{(2)} &= g(z_1^{(2)}) \\
a_2^{(2)} &= g(z_2^{(2)}) \\
a_3^{(2)} &= g(z_3^{(2)})
\end{aligned}
$$

We are multiplying our matrix $\Theta^{(j-1)}$ with dimensions $s_j \times (n+1)$ (where s_j is the number of our activation nodes) by our vector $a^{(j-1)}$ with height (n+1). This gives us our vector $z^{(j)}$ with height s_j. Now we can get a vector of our activation nodes for layer j as follows:

$$a^{(j)} = g(z^{(j)})$$

Where our function g can be applied element-wise to our vector $z^{(j)}$.

We can then add a bias unit (equal to 1) to layer j after we have computed $a^{(j)}$. This will be element $a_0^{(j)}$ and will be equal to 1. To compute our final hypothesis, let's first compute another z vector:

$$z^{(j+1)} = \Theta^{(j)} a^{(j)}$$

We get this final z vector by multiplying the next theta matrix after $\Theta^{(j-1)}$ with the values of all the activation nodes we just got. This last theta matrix $\Theta^{(j)}$ will have only **one row** which is multiplied by one column $a^{(j)}$ so that our result is a single number. We then get our final result with:

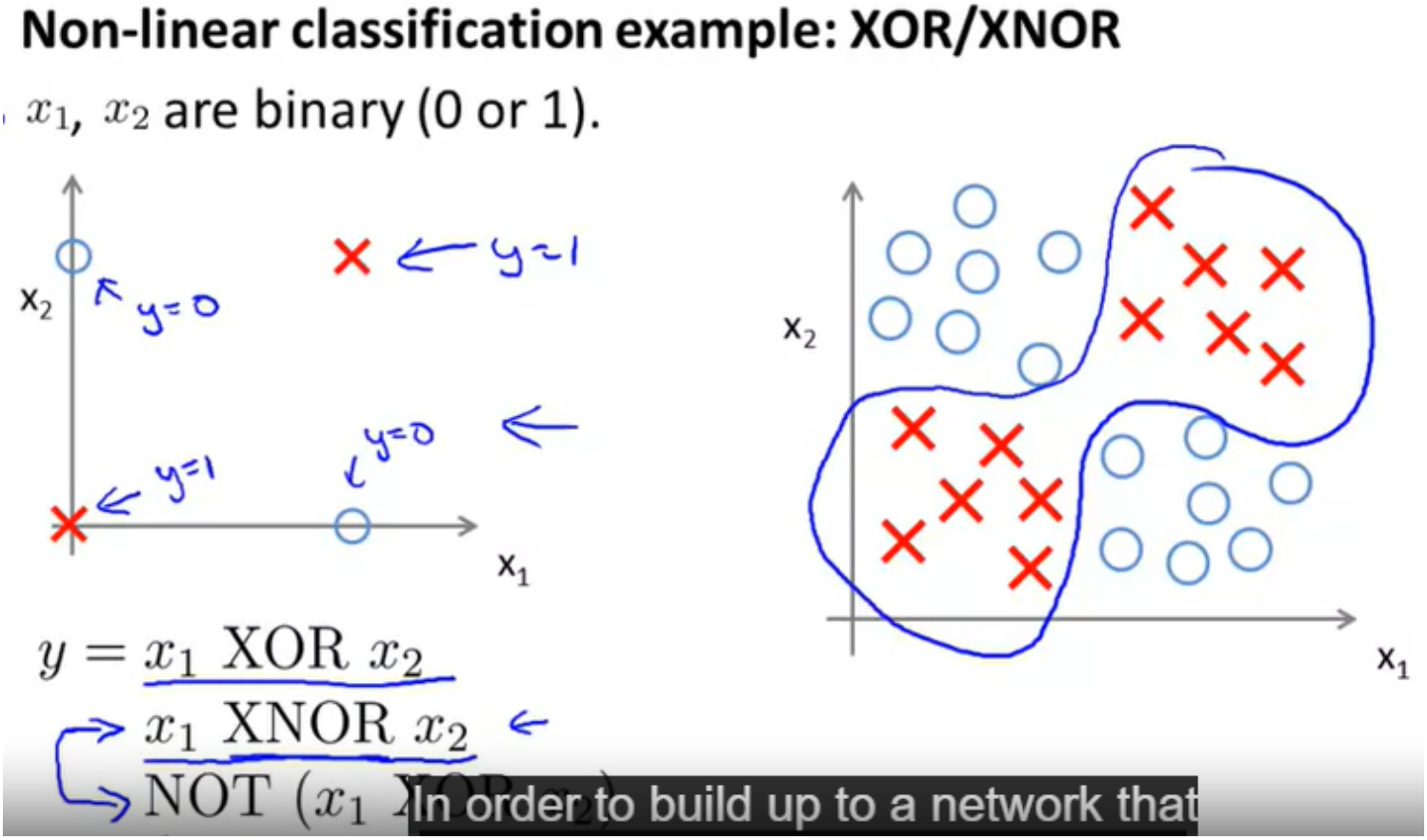
$$h_\Theta(x) = a^{(j+1)} = g(z^{(j+1)})$$

Notice that in this **last step**, between layer j and layer j+1, we are doing **exactly the same thing** as we did in logistic regression. Adding all these intermediate layers in neural networks allows us to more elegantly produce interesting and more complex non-linear hypotheses.

Applications

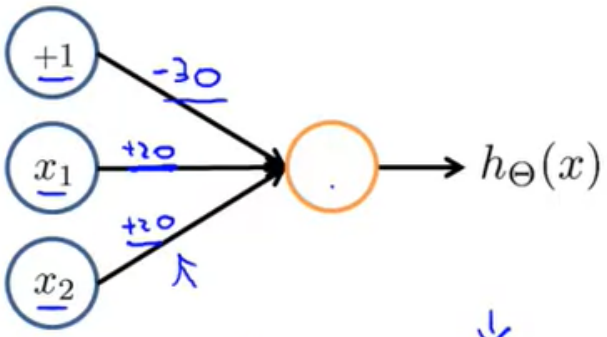
Examples And intuition I

non-linear decision boundary



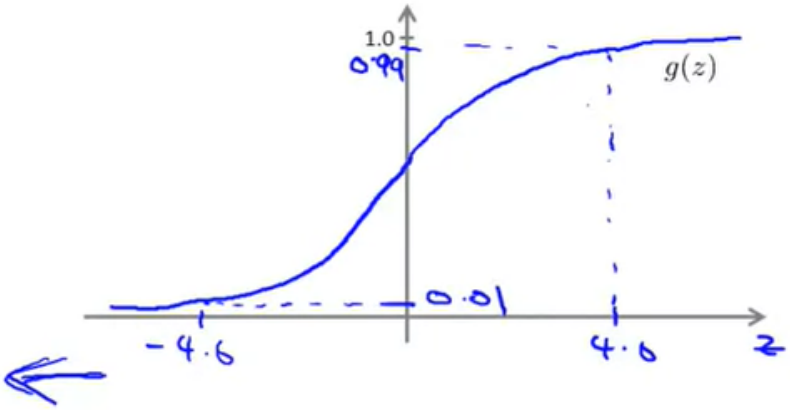
Simple example: AND

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1$ AND x_2



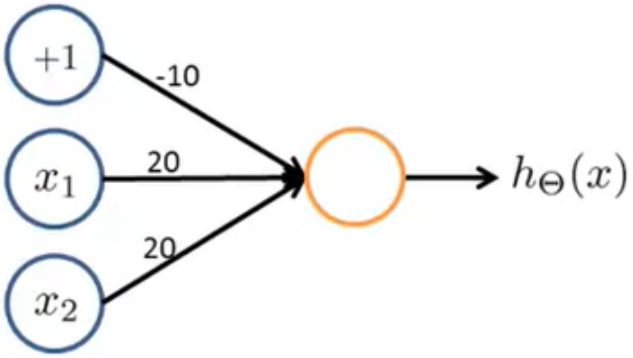
$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

$x_1^{(1)}$ $x_2^{(1)}$ $x_{12}^{(1)}$



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	$g(10) \approx 1$

Example: OR function

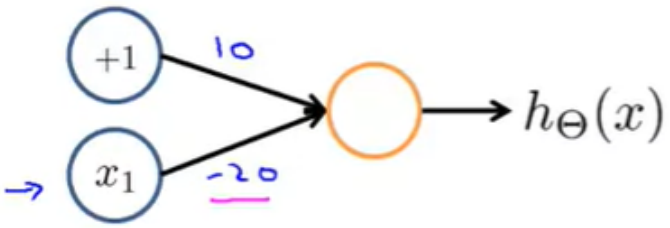


$g(-10 + 20x_1 + 20x_2)$

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

→ x_1 AND x_2 → x_1 OR x_2 {0,1}.

Negation: NOT x_1



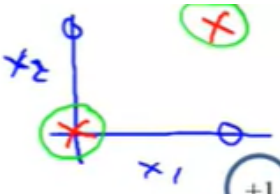
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

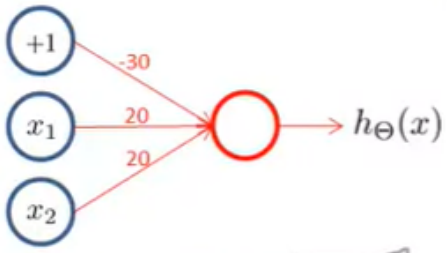
$h_\Theta(x) = g(10 - 20x_1)$

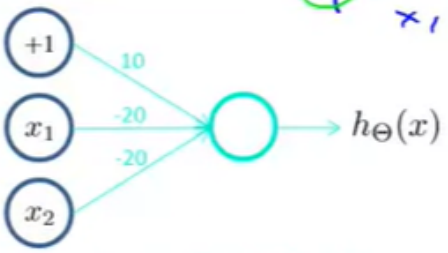
→ (NOT x_1) AND (NOT x_2)

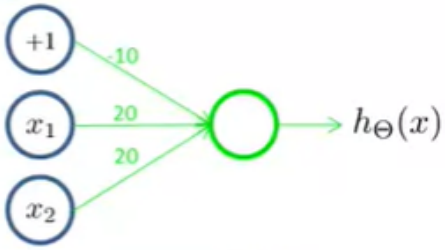
(=1 if and only if → $x_1 = x_2 = 0$

Putting it together: x_1 XNOR x_2

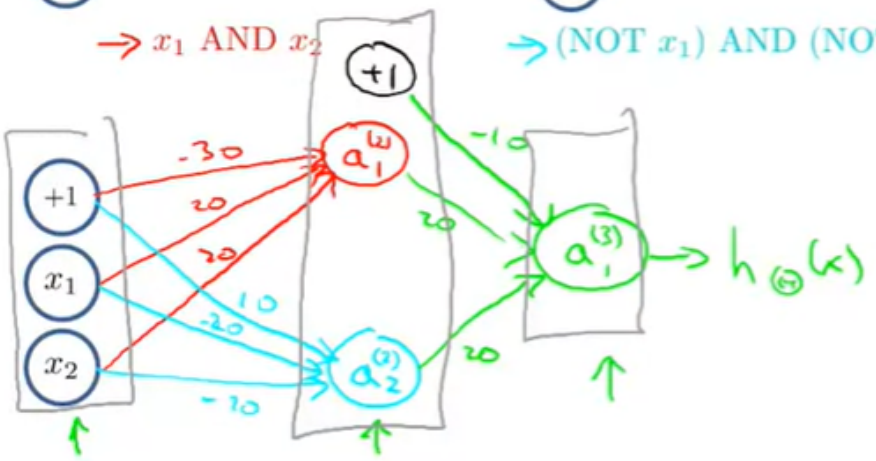






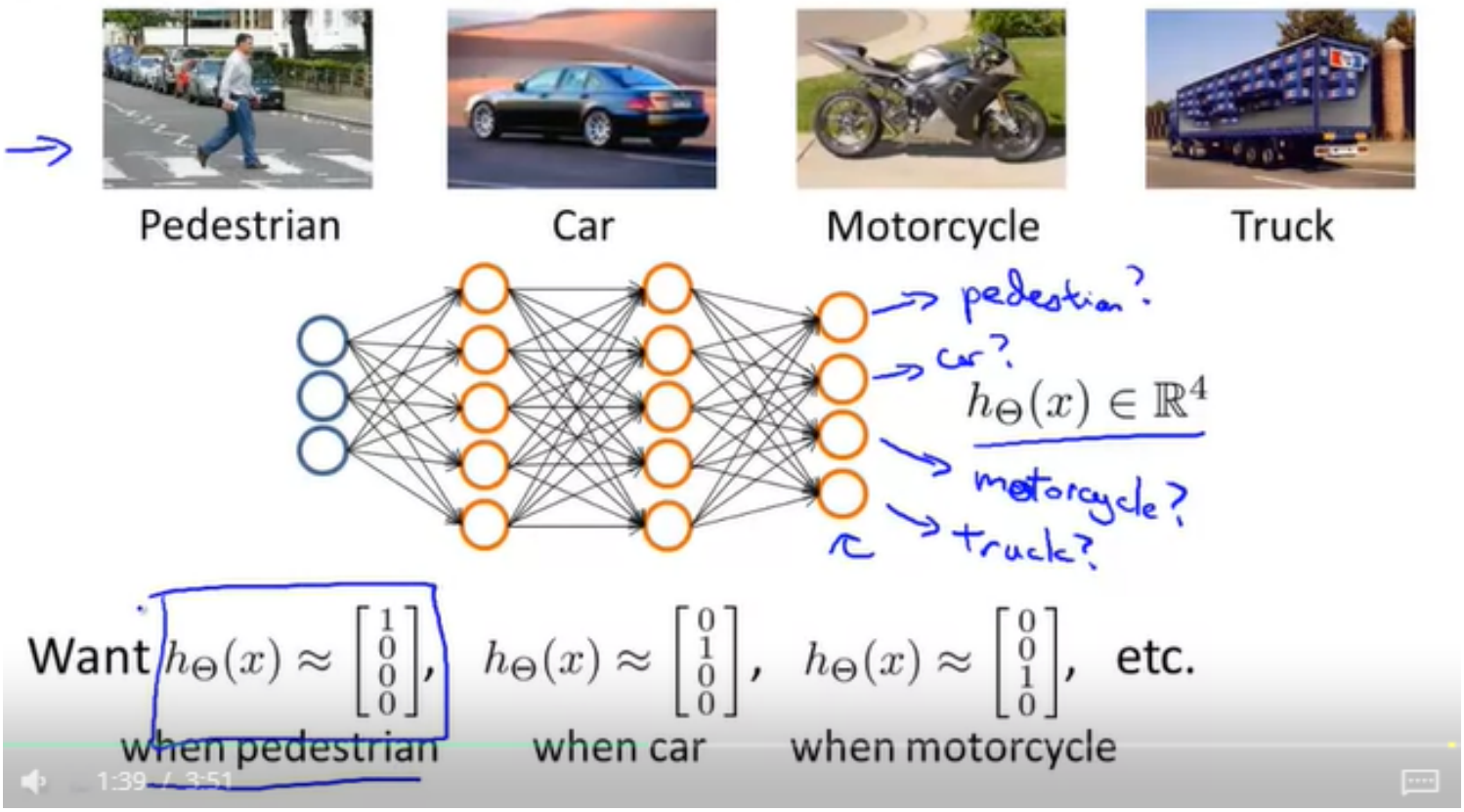


→ x_1 AND x_2 → (NOT x_1) AND (NOT x_2) → x_1 OR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
→ 0	0	0	1	1 ←
0	1	0	0	0
1	0	0	0	0
→ 1	1	1	0	1 ←

multiclass classification



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

Each $y^{(i)}$ represents a different image corresponding to either a car, pedestrian, truck, or motorcycle. The inner layers, each provide us with some new information which leads to our final hypothesis function. The setup looks like:

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

Our resulting hypothesis for one set of inputs may look like:

$$h_\Theta(x) = \begin{bmatrix}0010\end{bmatrix}$$

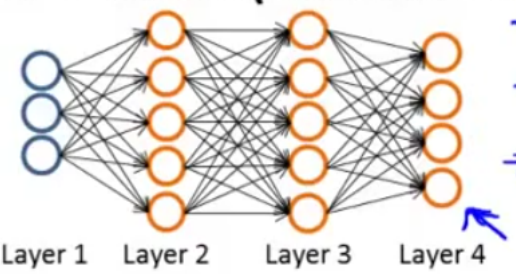
In which case our resulting class is the third one down, or $h_\Theta(x)_3$, which represents the motorcycle.

Week5

Neural network learning

application of nn in classification problem

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

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

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

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

Cost function
generalization of logistic regression
we do not regularise the biased term - 1...n

K=1..4 - 4 output units
summing the cost function on each of them

y(k) , h(k)

regularization term - lambda , for all values of i,j,l

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:

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

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

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

Andrev

algorithm the minimize the cost function
or backpropagation algorithm

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:

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

$\Theta_{ij}^{(l)} \in \mathbb{R}$

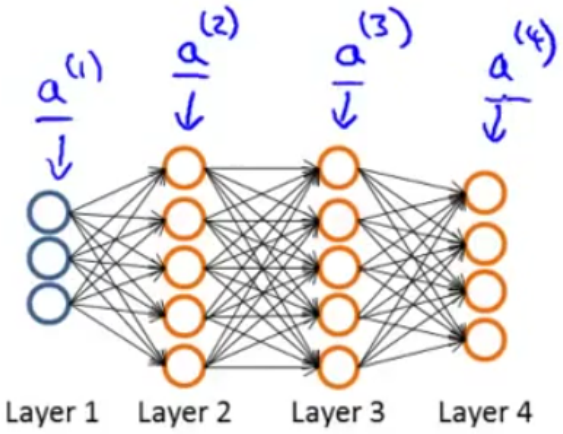
a(1) is the activation value or layer 1
z2 is required value for sigmoid function in the next layer

Gradient computation

Given one training example (x, y):

Forward propagation:

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



each node we compute a new delta j(l) is error of node j and layer l
L=4
for each layer we calculate the output unit

we have derivative of the sigmoid function in delta calculation

Gradient computation: Backpropagation algorithm

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

For each output unit (layer L = 4)

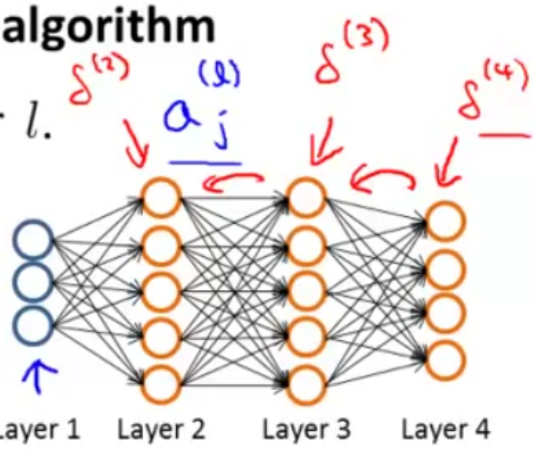
$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

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

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

(ignore λ; if $\lambda = 0$)



training set of m examples
traingle is capital delta

for ith iteration we are working with x(i),y(i)

perform forward propagation

activation calucalation

output layer - a(l) output of hypothesus layer minus target label was

we use the capital delta terms to accumulate the partial derivatives

we calculate seperate terms for j=0,j!=0

j=0 refers to the biased term taken in regard so no regularization terms

Backpropagation algorithm

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

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

For $i = 1$ to m

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

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

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

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

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

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

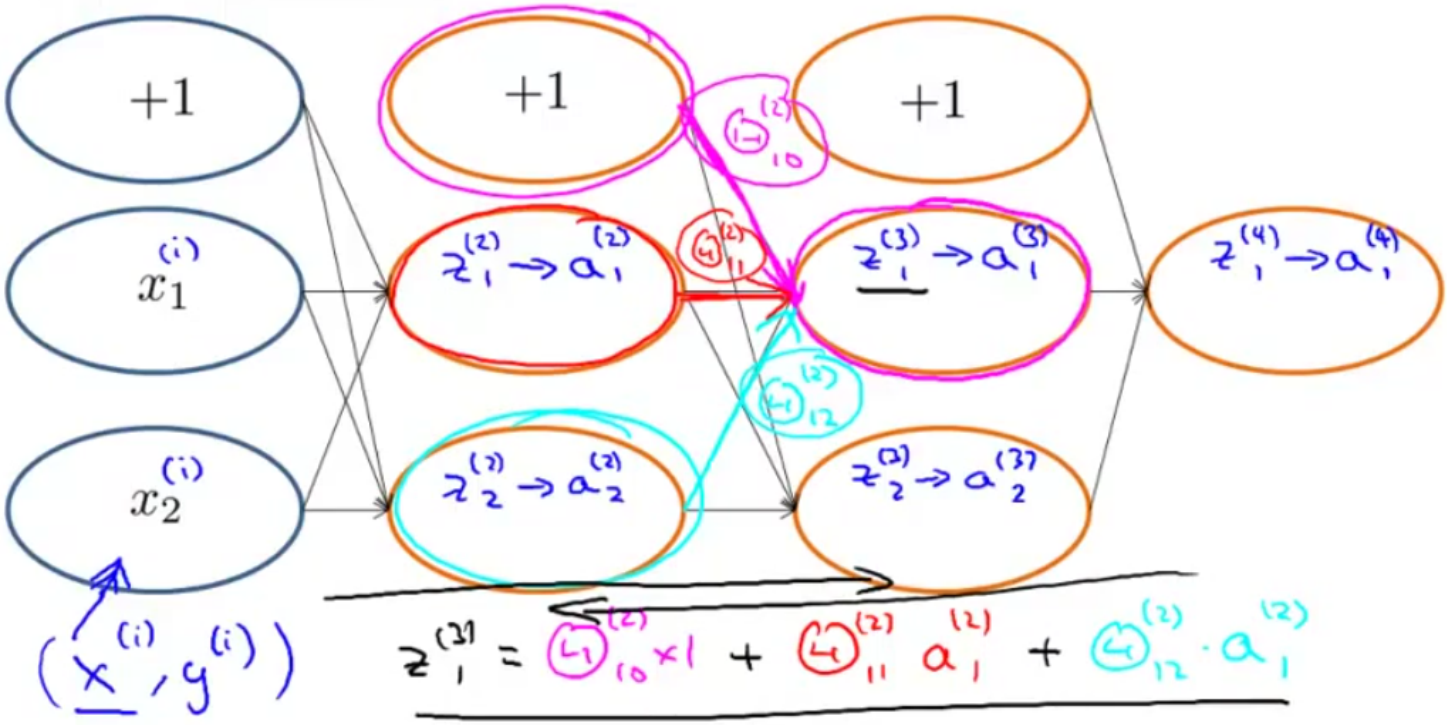
$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

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

Backpropagation Intuition

we use sigmoid function to get the activation values

Forward Propagation



beautifully explained the forward calculation over each layers and nodewise

cost term is associated with the ith term of the x(i),y(i)
cost(i) = (neural network value h(x) - actual value y(i))^2 to present only postive values

What is backpropagation doing?

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

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

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

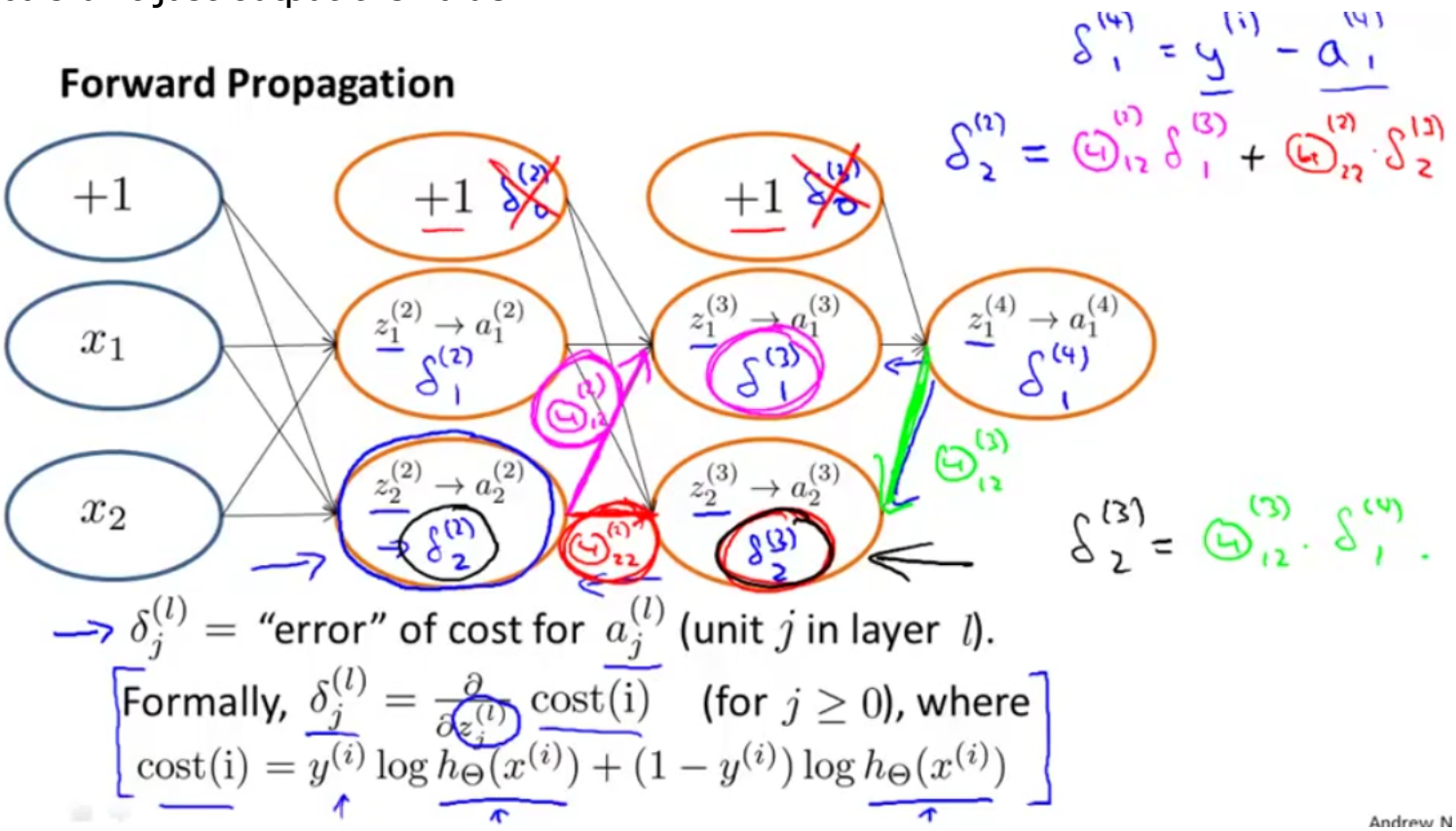
$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)
I.e. how well is the network doing on example i?

$y^{(i)}$

delta terms are partial derivatives of the cost function with respect to the neural network values of each layer of given specific example

delta value is calulated by multiply the edge weight with the delta value of the previous layer (D(i+1) = D(i)*theta(i+1)(i,i+1,...) + D(i)*theta(i+1)(i,i,...)
bais unit just output the value 1



implementation note: unrolling parameters

from matrixes to vectors
advance optimization

take input of pointed to cost function and some initial value of theta and options
this was fine for logistic regression
but in neural nets we have these matrices for theta terms, or gradient terms

so, we to present the matrixes into vectors so that they can be properly implemented in the optimization ~ Unroll the vectors

Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
  ...
optTheta = fminunc(@costFunction, initialTheta, options)
```

Neural Network (L=4):

→ $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)

→ $D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D1, D2, D3)

"Unroll" into vectors

s(i) = units in layer i

dimensions are expressed

all the elements of matrices will become one large vector

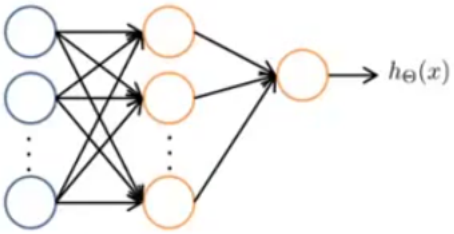
we use reshape with no of elements to pull out

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);
```

```
>> Theta1 = ones(10,11)
Theta1 =

   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
```

```
>> Theta2 = 2*ones(10,11)
Theta2 =

   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
```

```
>> thetaVec = [ Theta1(:); Theta2(:); Theta3(:) ];
>> size(thetaVec)
ans =

   231     1
```

```
>> reshape(thetaVec(1:110), 10,11)
ans =

   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1   1
```

```
>> reshape(thetaVec(111:220), 10,11)
ans =

   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2
   2   2   2   2   2   2   2   2   2   2   2

>> reshape(thetaVec(221:231), 1,11)
ans =

   3   3   3   3   3   3   3   3   3   3   3
```

implementation algo with required return value types as vectors

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 $J(\Theta)$.

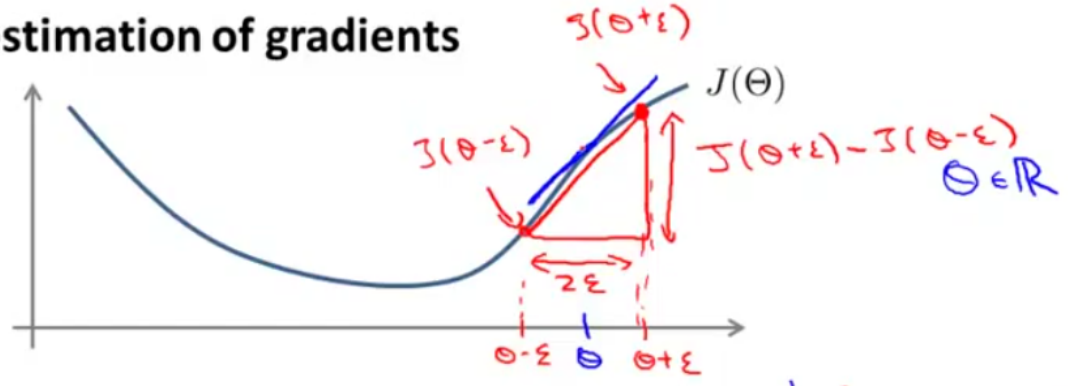
  Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get `gradientVec`.

Gradient Checking
it look like its working, cost function is decreasing with every implementation of gradient descent. there could be a bug that would create problems
so, we use gradient checking if there is any problem in the given training process for forward and backpropagation
verify the code is working

derivative - numerical process of take limit values for approximation of the slope/-tangent - derivative or gradient value
there are large value of episolon to be considered
if the value of episolon is taken very less then, it almost becomes the exact value of the paartial derivative of the J(theta)- there is now a numerical problem

one-sided difference gives less accuracy
two-sided gives better accuracy

Numerical estimation of gradients



$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon} \qquad \varepsilon = 10^{-4}$$

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

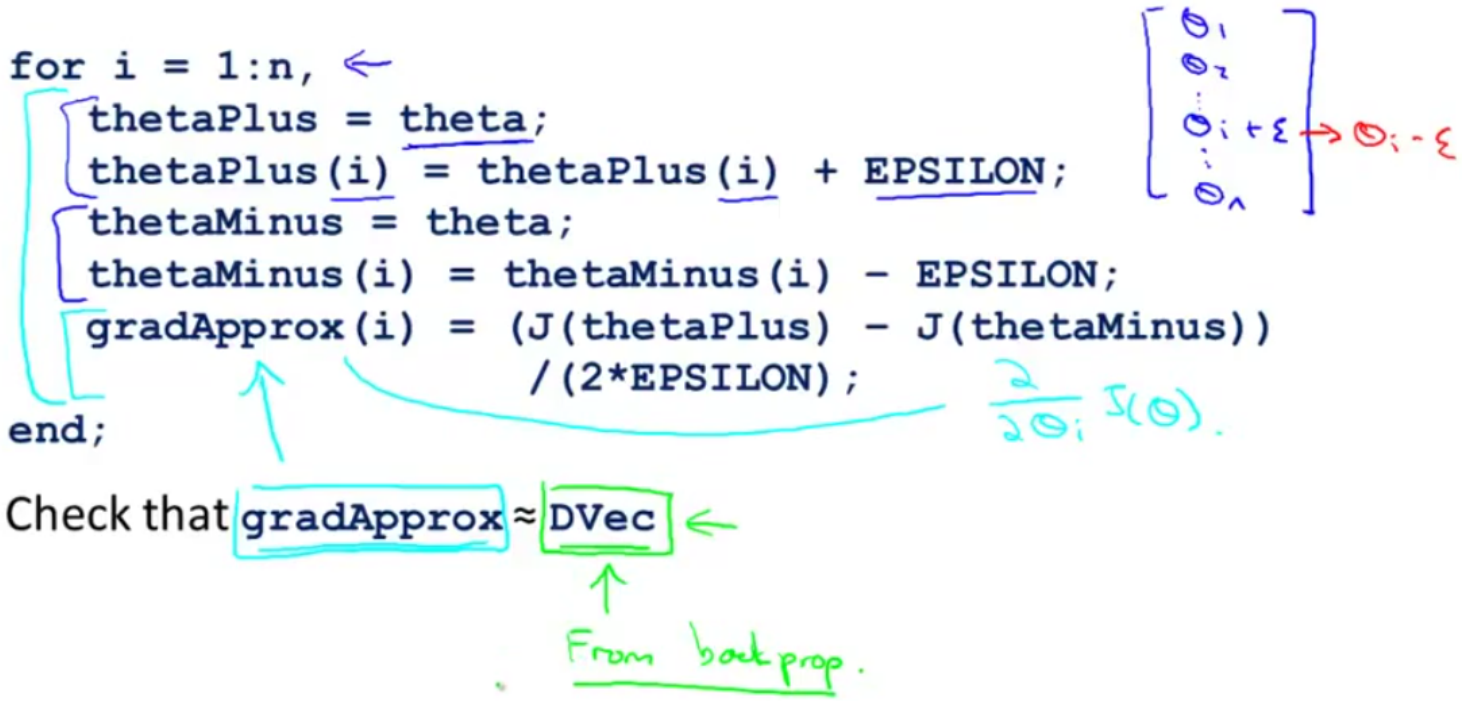
Implement: `gradApprox = (J(theta + EPSILON) - J(theta - EPSILON)) /(2*EPSILON)`

consider theta as vector

Parameter vector θ

- $\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)
- $\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$
- $\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon, \theta_2, \theta_3, \ldots, \theta_n) - J(\theta_1-\epsilon, \theta_2, \theta_3, \ldots, \theta_n)}{2\epsilon}$
- $\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \ldots, \theta_n)}{2\epsilon}$
- $\vdots$
- $\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n+\epsilon) - J(\theta_1, \theta_2, \theta_3, \ldots, \theta_n-\epsilon)}{2\epsilon}$

approximate the value of J wrt to theta(i)



once the implementation fo backprop is verified via gradient checking, we should stop using gradient checking else the code becomes slower

numerical gradient checking code is much slower than the back prop. algo

Implementation Note:

- Implement backprop to compute `DVec` (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$).
- Implement numerical gradient check to compute `gradApprox`.
- Make sure they give similar values.
- Turn off gradient checking. Using backprop code for learning.

Important:

- Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of `costFunction(...)`) your code will be very slow.

Random initialization

initializing the theta parameters zeros is okay with logistic regression
but doesnt work with NN
hidden layer would give same values of the function as the all the initial theta are same and zero

Initial value of Θ

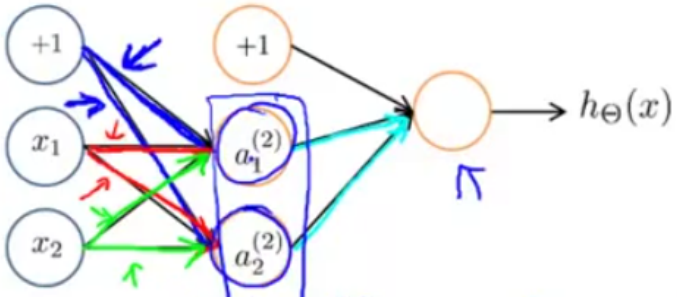
For gradient descent and advanced optimization method, need initial value for Θ.

```
optTheta = fminunc(@costFunction,
           initialTheta, options)
```

Consider gradient descent

Set `initialTheta = zeros(n,1)` ?

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.

$a_1^{(2)} = a_2^{(2)}$

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;
```

$[-\epsilon, \epsilon]$

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

Putting it together

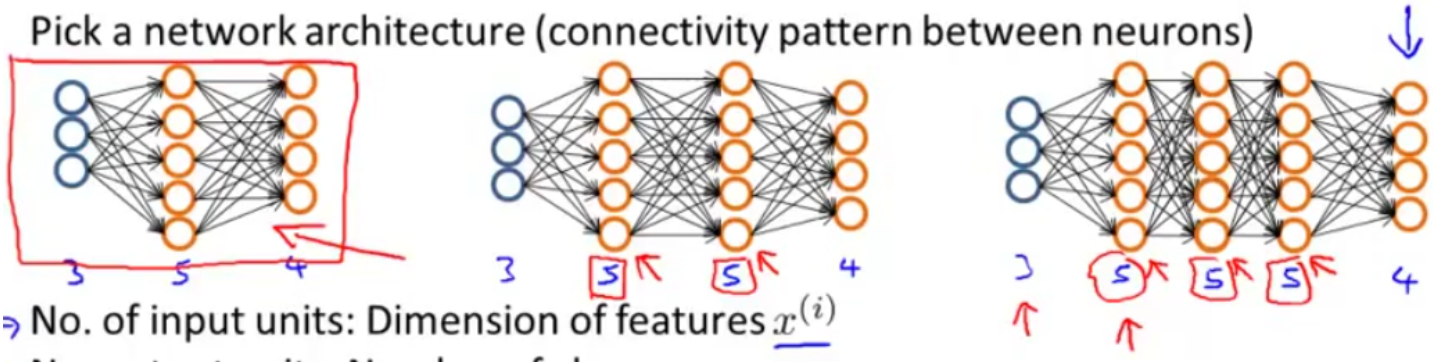
choices are archietectural choices

no of input units : dimensions of features
no of output units : no of classes : y : 1,2,3.....K

same no units in every single hidden layer ?
increasing the number of hidden layers makes it more computationaly expensive - more time - slower

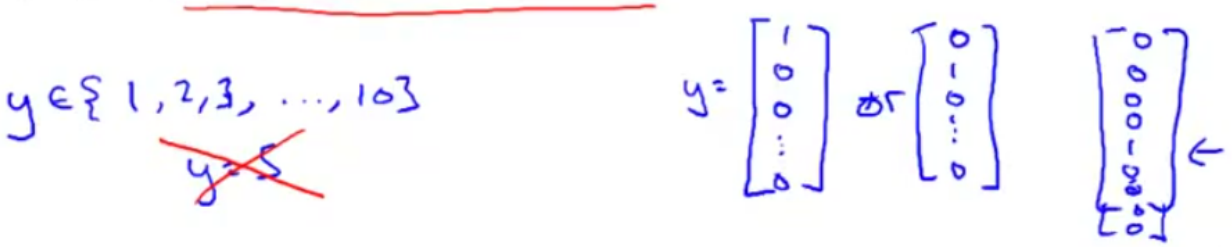
Training a neural network

Pick a network architecture (connectivity pattern between neurons)



- No. of input units: Dimension of features $x^{(i)}$
- No. output units: Number of classes

Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)



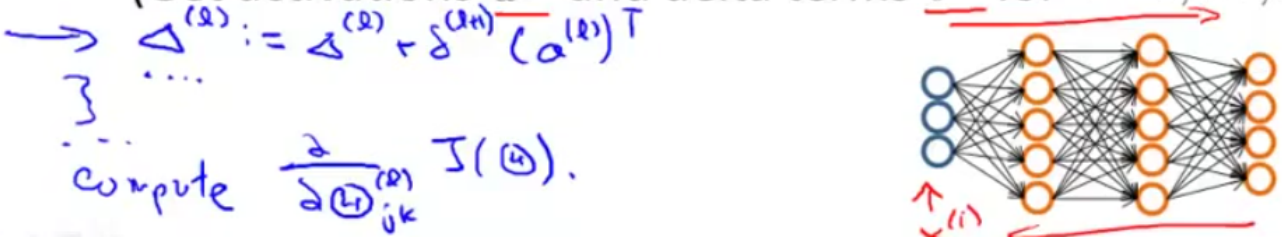
backprop - for loop to iterate over the examples
for first time

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 { $(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$, ... $(x^{(m)}, y^{(m)})$

Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).



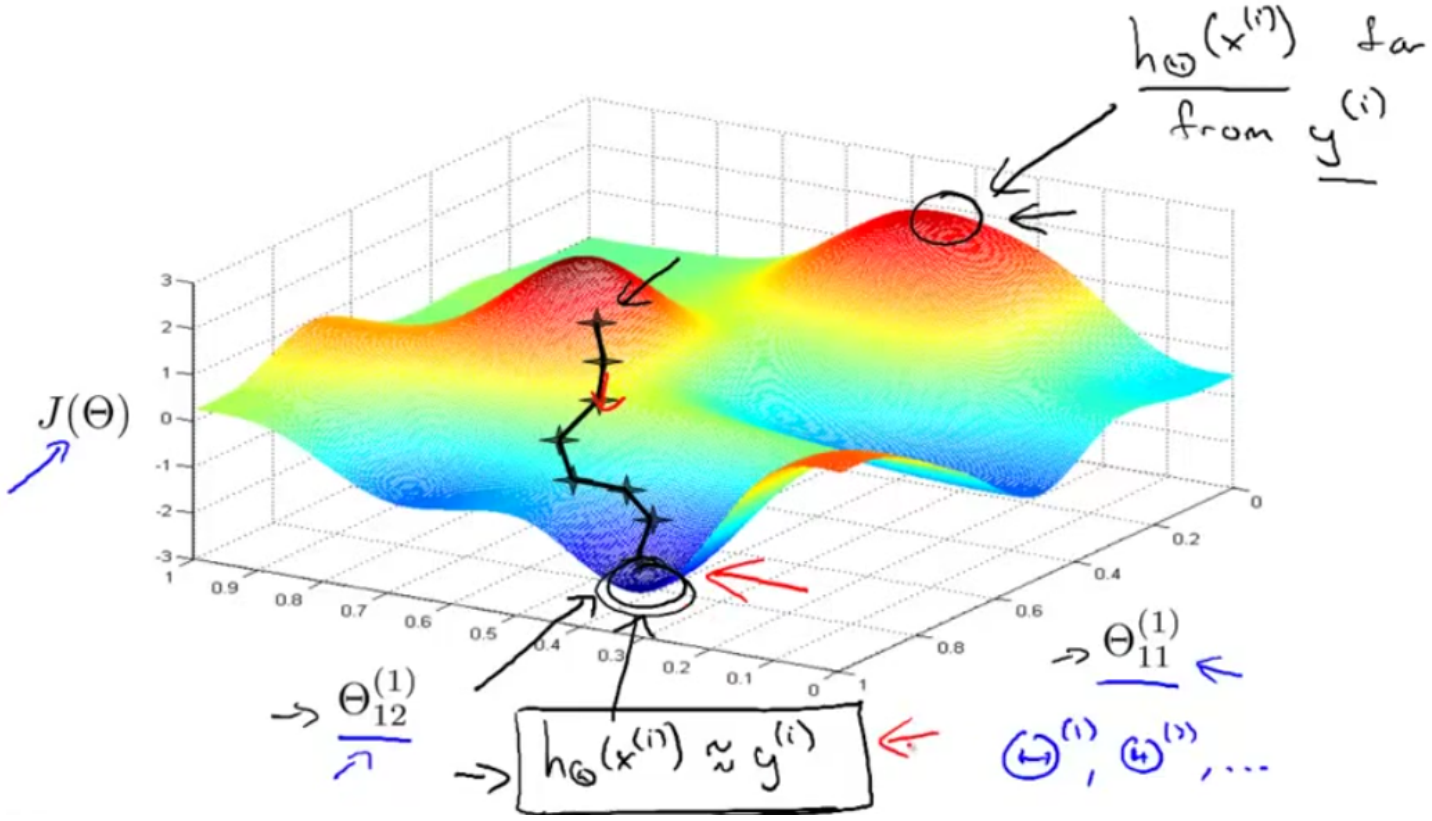
advance optimization - can get stuck in local optima
usually gradient descent works fine

Training a neural network

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.
   - Then disable gradient checking code.
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

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

$J(\Theta)$ — non-convex.



Autonomous Driving
- getting a car to learn to drive

direction of selected by human driver
left right
NN direction selected
left right

caamera view of the car

Week6

Advice for applying Machine Learning

Deciding what to try next

really know how to know to implement the algorithms
make sure the ml systems, to the most promising avenues to invest time
improve the ML system

getting more data might not be so helpful always
trying to use smaller set of features to avoid overfitting
trying getting additional features
trying adding polynomial features

Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices.

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

However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- Get more training examples
- Try smaller sets of features $x_1, x_2, x_3, \ldots, x_{100}$
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc.})$
- Try decreasing λ
- Try increasing λ

Machine learning diagnostic:

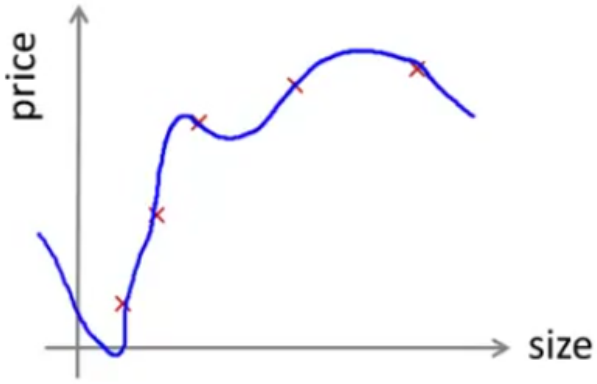
Diagnostic: A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

Diagnostics can take time to implement, but doing so can be a very good use of your time.

Evaluating A hypothesis
prevent the problems of overfitting and underfitting

low training error doesnt mean better hypothesis
failing to generalize the function

Evaluating your hypothesis



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Fails to generalize to new examples not in training set.

x_1 = size of house
x_2 = no. of bedrooms
x_3 = no. of floors
x_4 = age of house
x_5 = average income in neighborhood
x_6 = kitchen size
$\vdots$
x_{100}

split the data into two portions
i.e 70/30 split

Evaluating your hypothesis

Dataset:

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

70% Training set

30% Test set

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

$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
$\vdots$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = no. of test example

$(x_{test}^{(i)}, y_{test}^{(i)})$

learn the parameter from train data

compute test set error

Training/testing procedure for logistic regression

- Learn parameter θ from training data
- 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):

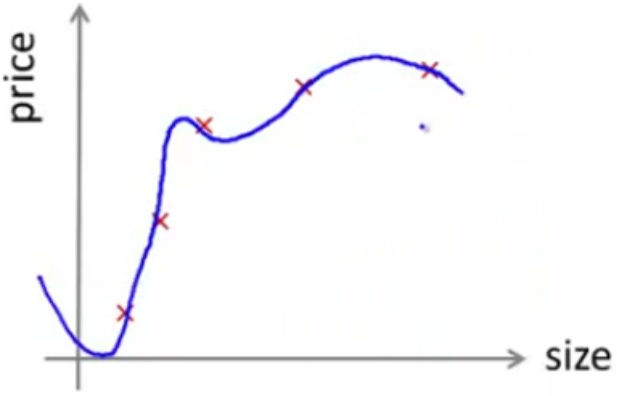
$$err(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, \ y = 0 \text{ or if } h_\theta(x) < 0.5, \ y = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{error}$$

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^{(i)}), y_{test}^{(i)})$$

Model Selection and training/validation/test_sets
What ?
degree of polynomial to consider
which features to consider
choose the regularization lambda value for learning algorithms

these are model selection problem

Overfitting example



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Once parameters $\theta_0, \theta_1, \ldots, \theta_4$ were fit to some set of data (training set), the error of the parameters as measured on that data (the training error $J(\theta)$) is likely to be lower than the actual generalization error.

choose a model that wouldfit the model, also estimate the model gives fitted output overly optimistic generalization layer

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 $J_{test}(\theta^{(5)})$.

$\theta_0, \theta_1 \ldots$

$\theta^{(5)}$

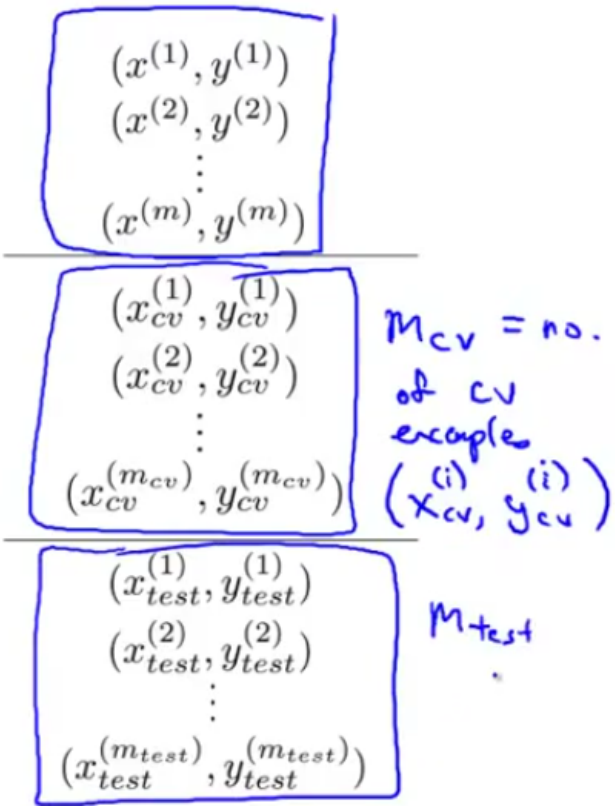
Problem: $J_{test}(\theta^{(5)})$ is likely to be an optimistic estimate of generalization error. I.e. our extra parameter (d = degree of polynomial) is fit to test set.

what is cross-validation purpose? ~validation set

Evaluating your hypothesis

Dataset:

Size	Price	
2104	400	Training set (60%)
1600	330	
2400	369	
1416	232	
3000	540	
1985	300	
1534	315	Cross validation set (CV) (20%)
1427	199	
1380	212	test set (20%)
1494	243	



Train/validation/test error

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:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

instead of using test set for model selection, we use validation set for model selection

pick the lowest vaalidation error

Model selection

1. $h_\theta(x) = \theta_0 + \theta_1 x$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ → $\theta^{(2)}$ → $J_{cv}(\theta^{(2)})$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$ → $\theta^{(3)}$ → $J_{cv}(\theta^{(3)})$

$\vdots$

10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$ → $\theta^{(10)}$ → $J_{cv}(\theta^{(10)})$

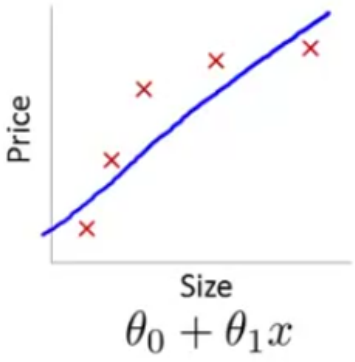
$d = 4$

Pick $\theta_0 + \theta_1 x_1 + \cdots + \theta_4 x^4$ ←

Estimate generalization error for test set $J_{test}(\theta^{(4)})$ ←

Diagnosing bias vs Variance
underfitting or overfitting problem

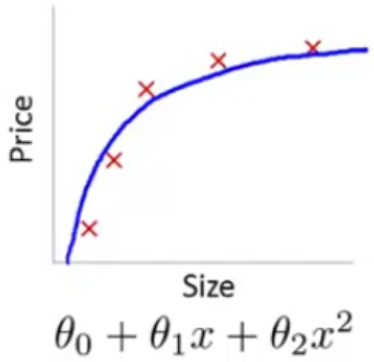
Bias/variance



$\theta_0 + \theta_1 x$

High bias (underfit)

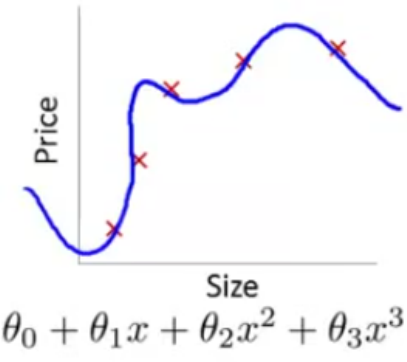
$d = 1$



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

"Just right"

$d = 2$



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

High variance (overfit)

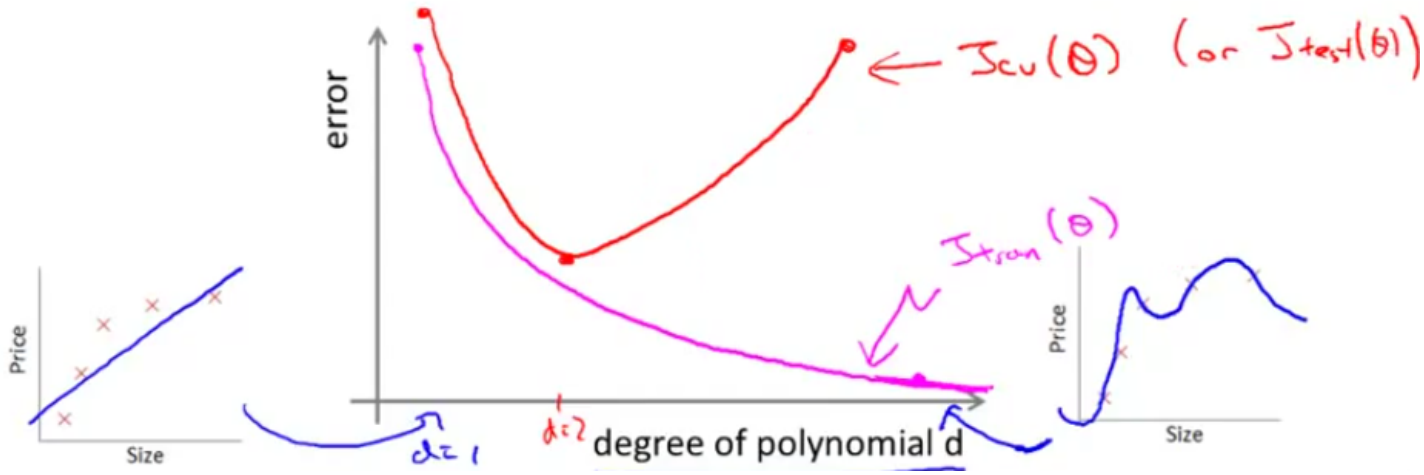
$d = 4$

given below of training error graph. the error steeps more quickly than depicted

Bias/variance

Training error: $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

Cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$ (or $J_{test}(\theta)$)

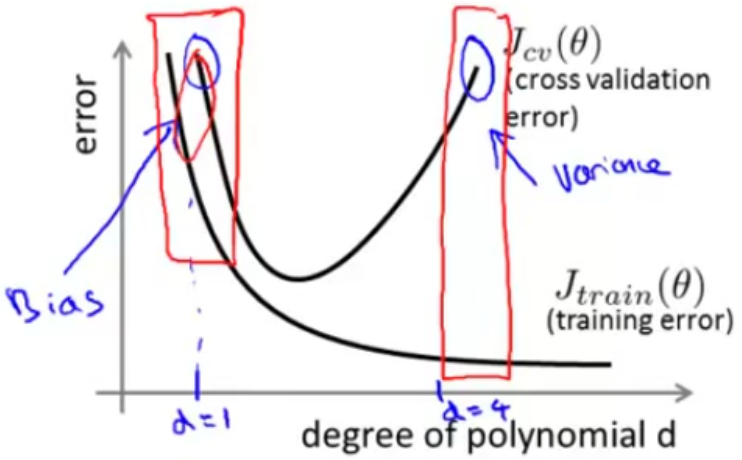


my doubt is how to select the order of features? coz each linear function with different feaature would give different slope of error, and considering a set of features would be tidious first hand.
now in error and validation error graph. the curve of cross validation could give more than one minima or what?

low order polynomial - high bias
high order polynomial - high variance

Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):
→ $J_{train}(\theta)$ will be high
$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):
→ $J_{train}(\theta)$ will be low
$J_{cv}(\theta) \gg J_{train}(\theta)$

>>

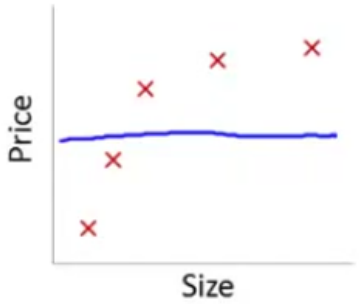
Regularization and Bias/Variance

effect of regularization on bias/variaance

Linear regression with regularization

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

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

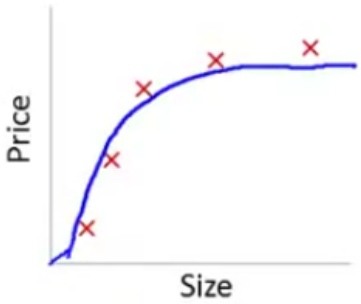


Large λ

High bias (underfit)

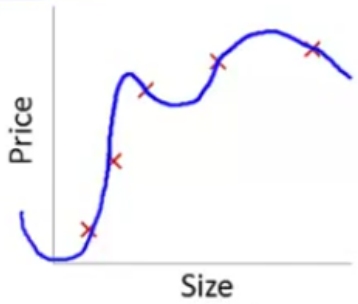
$\lambda = 10000.\ \theta_1 \approx 0, \theta_2 \approx 0, \ldots$

$h_\theta(x) \approx \theta_0$



Intermediate λ

"Just right"



Small λ

High variance (overfit)

$\lambda = 0$

Choosing the regularization parameter λ

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

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

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

take all the vaalues and put them on cross validation set, based on average sq error
pick any one which gives lowest cv error value
take the paramater and test how it works on test set

Choosing the regularization parameter λ

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 $\lambda = 0.08$

⋮

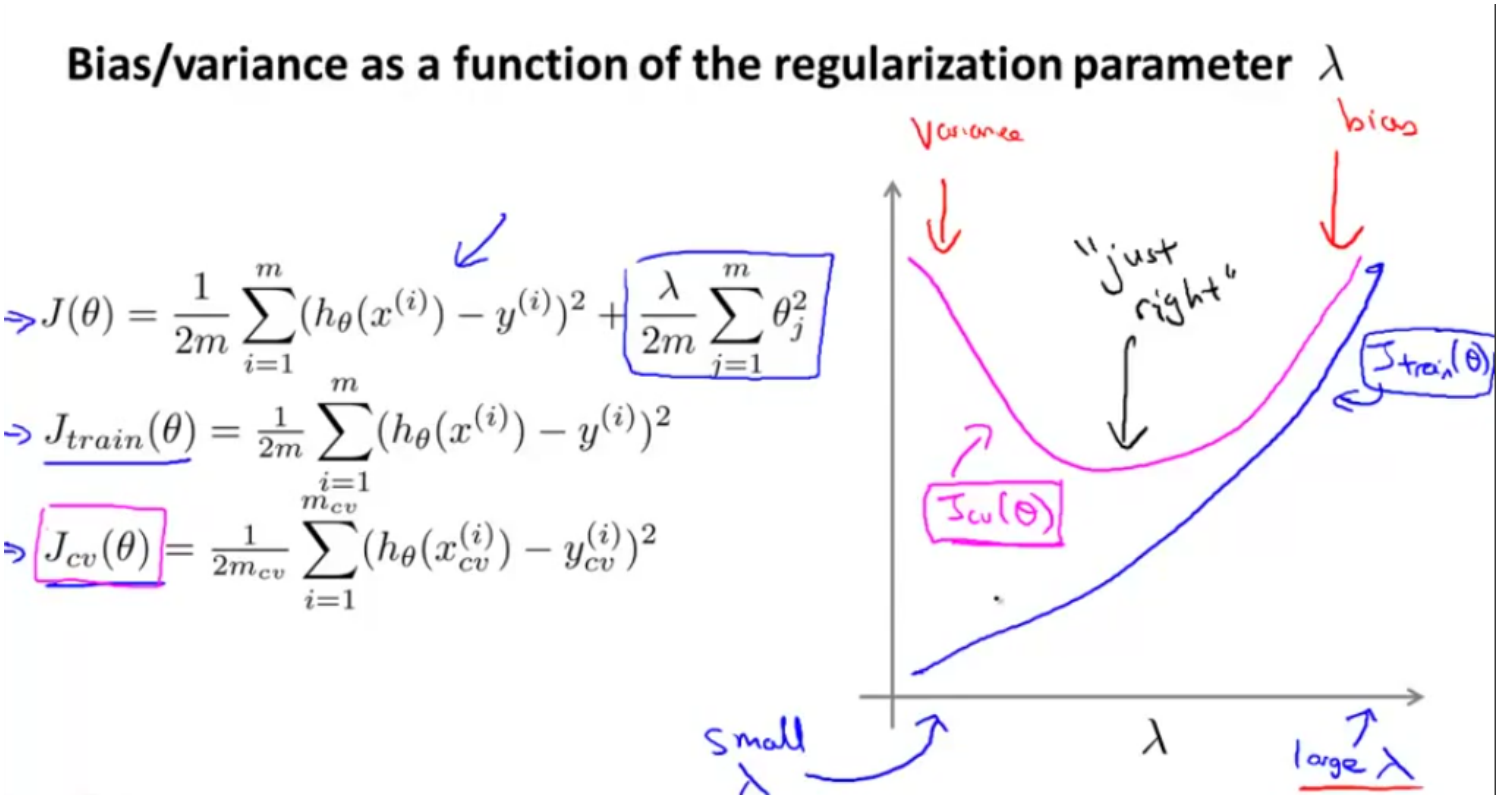
12. Try $\lambda = 10$

Pick (say) $\theta^{(5)}$. Test error: $J_{test}(\theta^{(5)})$

compare the J train and J cv

small lambda, not using regularization
large lambda- problem of bias - use regularization

bais-underfitting
on cv we find better variance and bias



on real problems, we could find similar trends, but we can find alot of noise sometimes

Learning curves
sanity check of algorithms, diagonizing the probelm
plot J(train) or J(cv) as a function of m (training examples)
artificially reduce set ize
limit the training examples

m = 1,2,3 then we have error = zero w/out regularization

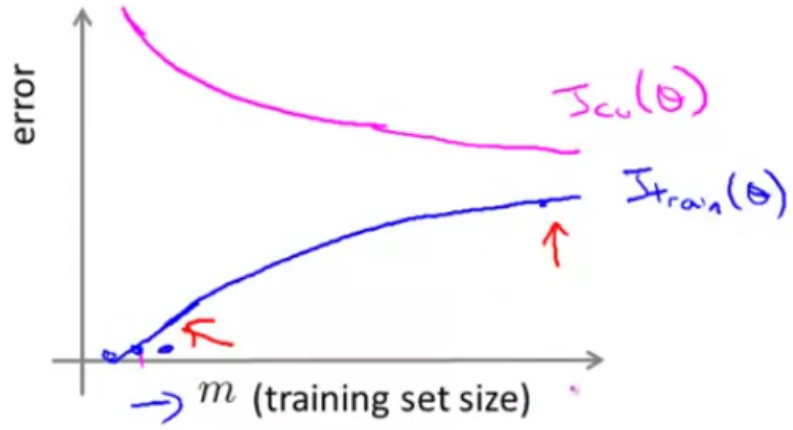
= ~zero w/ regularization

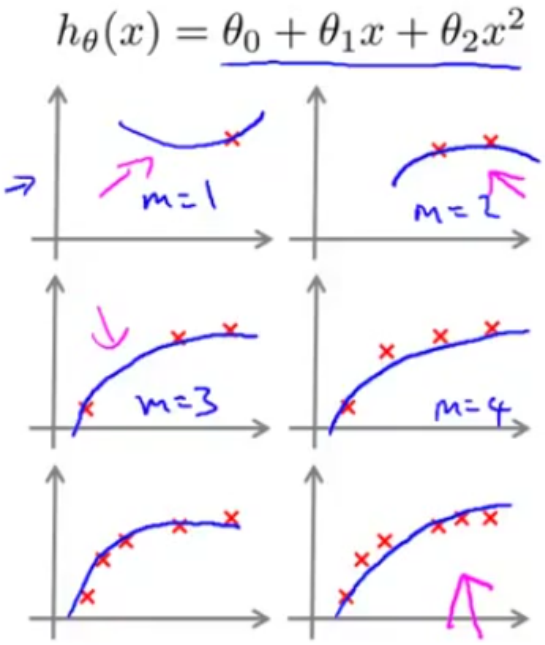
smaller the training set, smaller error
average training set error grows with more training set size

Learning curves

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

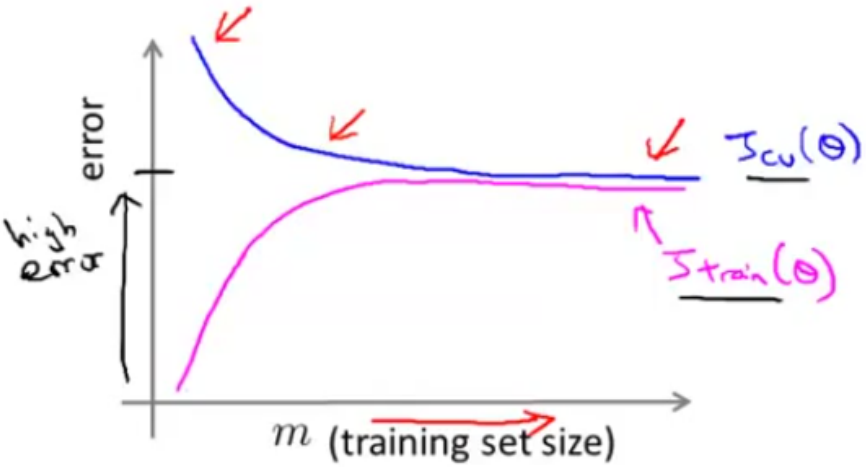
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

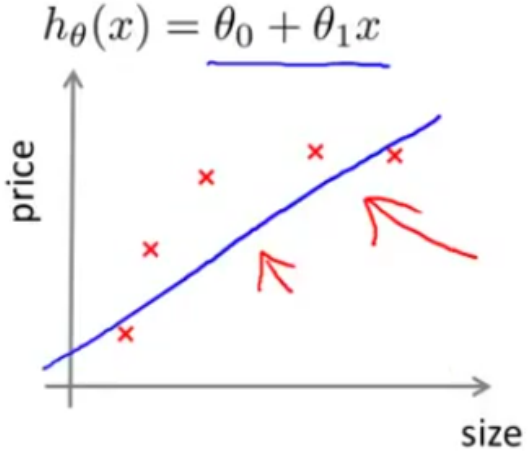


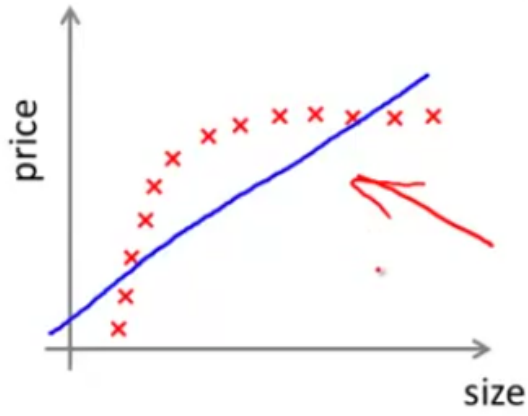


straight line is pretty much the same
high bias, the Jtrain will come close to Jcv
problem is high error is persistent

High bias



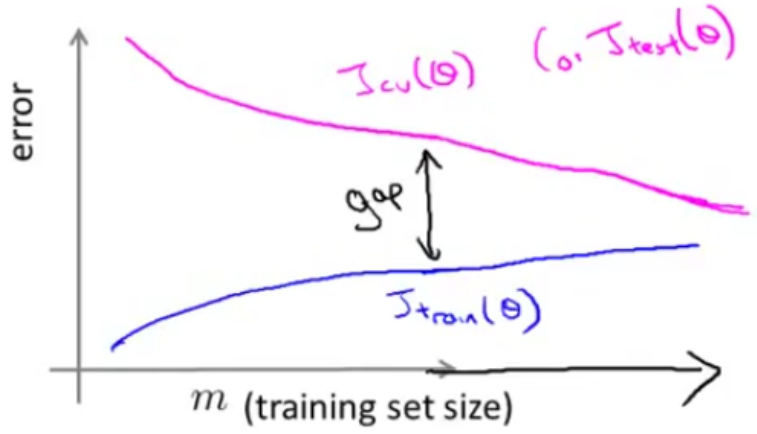




If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

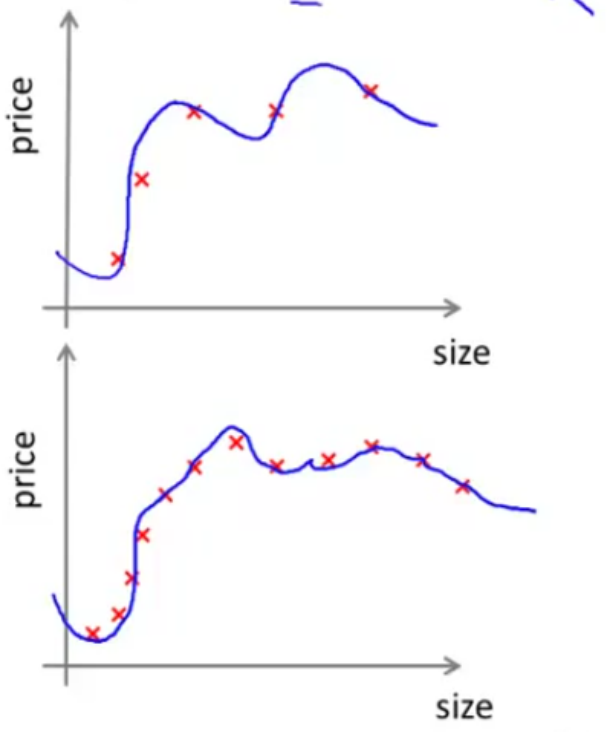
J train will lower
J cv will remain higher to high
there is a large gap

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) ↖



Deciding what to do next revisited

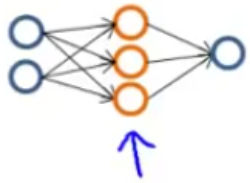
Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- 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 $(x_1^2, x_2^2, x_1 x_2, \text{etc})$ → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

regularization is good for large NN

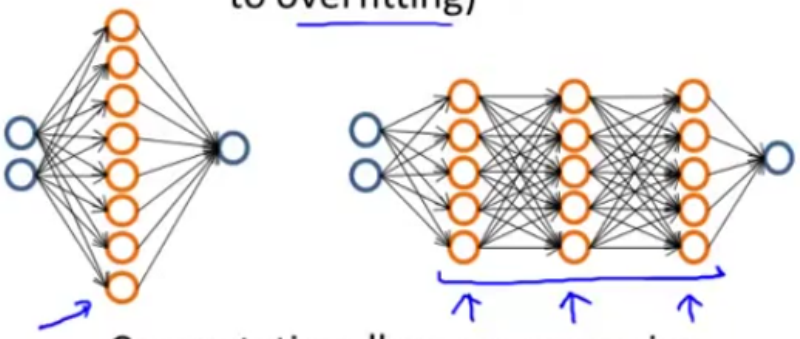
Neural networks and overfitting

"Small" neural network (fewer parameters; more prone to underfitting)



Computationally cheaper

"Large" neural network (more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization (λ) to address overfitting.

$J_{c_0}(\theta)$

Machine Learning System Design

Prioritizing What to work on : Spam classification example

Building a spam classifier

```
From: cheapsales@buystufffromme.com
To: ang@cs.stanford.edu
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - $50
Also low cost M0rgages
available.
```

Spam (1)

```
From: Alfred Ng
To: ang@cs.stanford.edu
Subject: Christmas dates?

Hey Andrew,
Was talking to Mom about plans
for Xmas. When do you get off
work. Meet Dec 22?
Alf
```

Non-spam (0)

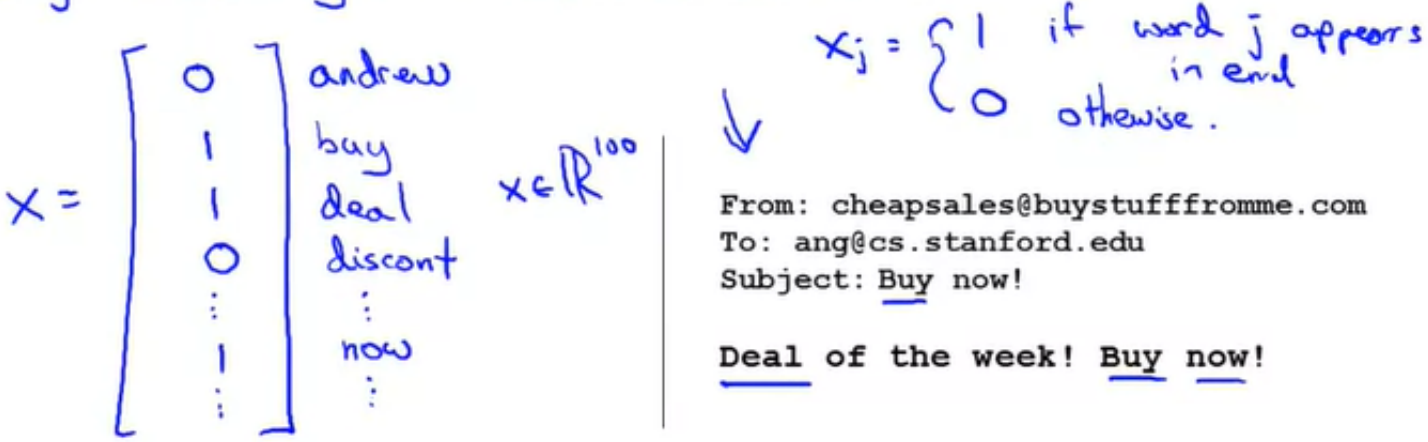
building a spam classifier :
given x and y we caan train using logistic regression

Building a spam classifier

Supervised learning. x = features of email. y = spam (1) or not spam (0).

Features x: Choose 100 words indicative of spam/not spam.

E.g. deal, buy, discont, andrew, now, ...



Note: In practice, take most frequently occurring n words (10,000 to 50,000) in training set, rather than manually pick 100 words.

Building a spam classifier

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.)

Error Analysis

helps in avoid premature optimization

Recommended approach

- Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

type and features considered are important for analysis hereS

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: 5 (m0rgage, med1cine, etc.)

Unusual email routing: 16

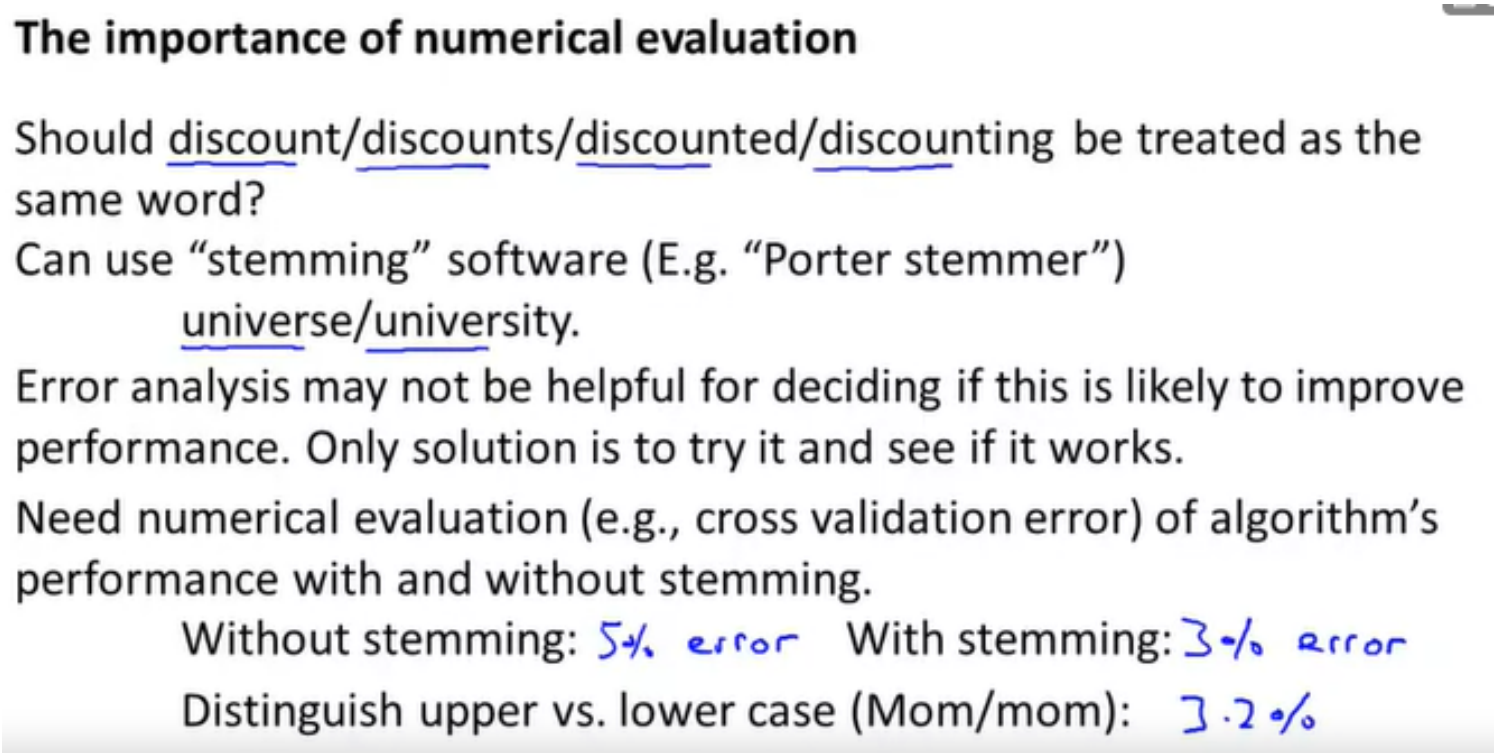
Unusual (spamming) punctuation: 32

numerical evaluation of learning algo

Why is the recommended approach to perform error analysis using the cross validation data used to compute $J_{CV}(\theta)$ rather than the test data used to compute $J_{test}(\theta)$?

- The cross validation data set is usually large.
- This process will give a lower error on the test set.
- If we develop new features by examining the test set, then we may end up choosing features that work well specifically for the test set, so Jtest(θ) is no longer a good estimate of how well we generalize to new examples.
- Doing so is less likely to lead to choosing an excessive number of features.

stemming - takes the words of verb form/pluralform/continuous tense form/ as one word

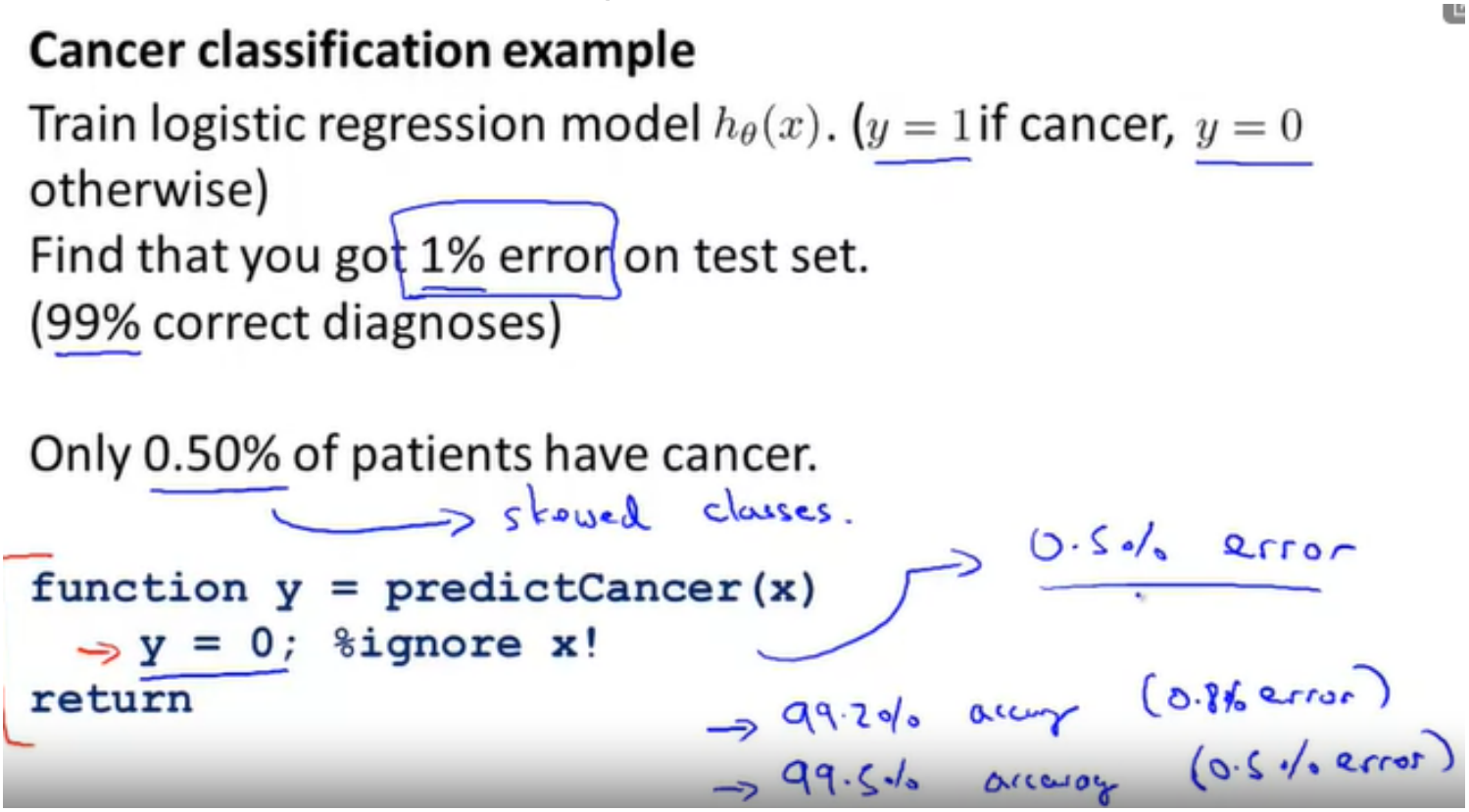


error analysis on cv set rather that test set
It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess your algorithm's performance. For example if we use stemming, which is the process of treating the same word with different forms (fail/failing/-failed) as one word (fail), and get a 3% error rate instead of 5%, then we should definitely add it to our model. However, if we try to distinguish between upper case and lower case letters and end up getting a 3.2% error rate instead of 3%, then we should avoid using this new feature. Hence, we should try new things, get a numerical value for our error rate, and based on our result decide whether we want to keep the new feature or not.

Handling Skewed Data

error metrics for skewed classes

in the below image, we see the case of positive example is much much smaller than the number of negative examples beacuse y equals one so rarely;
this is what a skewed class is.
lot more examples from one class than other class
it becomes much harder to use just clasification accuracy, becuase you can get very high classification accurcies or very low errors nd its not always clear if doing so is really improving the quality of classifier. So to get a clear image on this, we need a different error metrics; like precision/recall



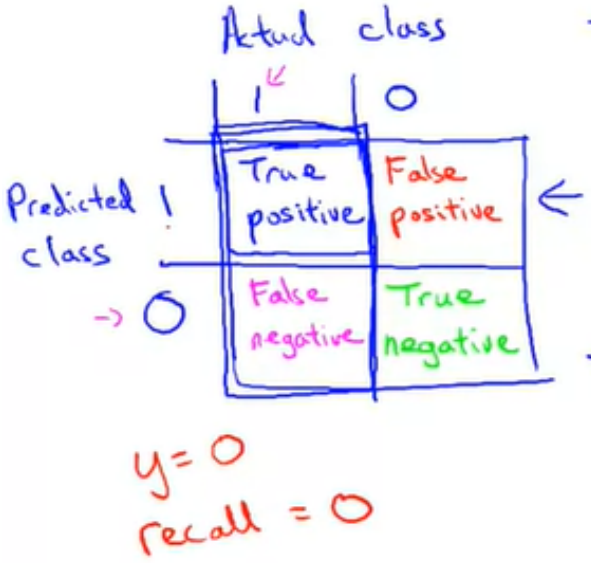
True positive
True negative
False positive
False negative

precision
total number in the first row are the predicted class true y=1
recall - of all the patients that actually have cancer, how many we have predicted have cancer

this will give a indication of how our classifier is doing

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**

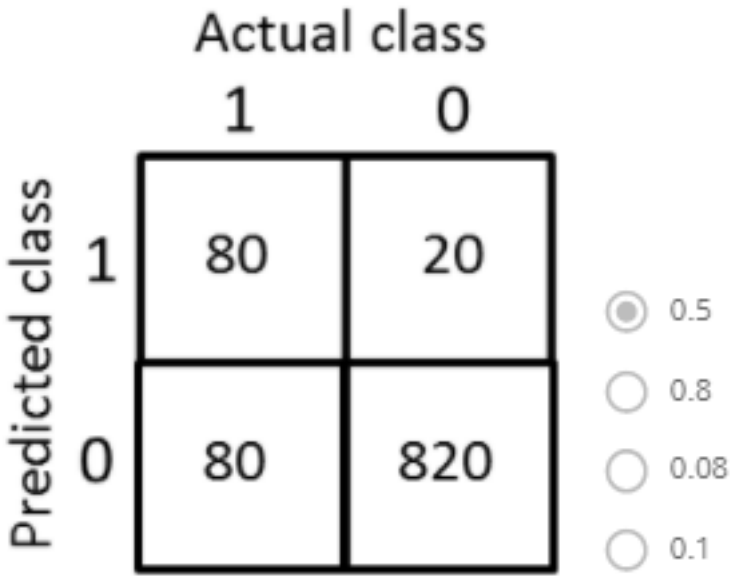
(Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$\frac{\text{True positives}}{\#\text{actual positives}} = \frac{\text{True positives}}{\text{True pos + False neg}}$$

Your algorithm's performance on the test set is given to the right. What is the algorithm's precision?

		Actual class 1	Actual class 0
Predicted class	1	80	20
	0	80	820

precision



recall

precision and recall are defined setting y equals 1, rather than y equals 0, to be sort of that the presence of that rare
class that we're trying to detect. And in particular, if a classifier is getting high precision and high recall, then we are actually confident that the algorithm has to be doing well, even if we have very skewed classes. So for the problem of skewed classes precisionrecall gives us more direct insight into how the learning algorithm is doing and this is often a much better way to evaluate our learning algorithms, than looking at classification error or classification accuracy, when the classes are very skewed.

Trading off precision and recall

Choosing higher accuracy rate as .9 or .7,this will affect the outcome we wish to find. so this would be a higher precision classifier will have lower recall because we want to correctly detect that those patients have cancer
another problem is to avoid too many false negatives. the above approach may seem to solve problem but the false negatives are really not an option to skip there is a doubt , so it better to report it as cancer. so we might instead set the value to lower level like .3 and be conservative that appropriate to seek for treatment. so this would be a higher recall classifier and going to be flaggin a higher fraction of all of the patients that actually do have cancer

Trading off precision and recall

→ precision = $\frac{\text{true positives}}{\text{no. of predicted positive}}$

→ recall = $\frac{\text{true positives}}{\text{no. of actual positive}}$

→ Logistic regression: $0 \le h_\theta(x) \le 1$

Predict 1 if $h_\theta(x) \ge 0.5$ ~~0.7~~ ~~0.9~~ 0.3

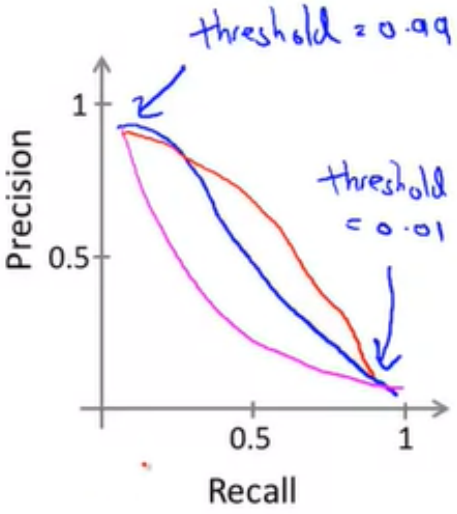
Predict 0 if $h_\theta(x) < 0.5$ ~~0.7~~ ~~0.9~~ 0.3

→ 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.



More generally: Predict 1 if $h_\theta(x) \ge$ threshold.

the precision recall curve can look very different shapes

Is there a way to choose this values automatically ? how to decide what is best among different approachs

taking average is not so good for any algo

so we look at F1 score - generally used in ML

◉ Measure precision (P) and recall (R) on the **cross validation set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$

F_1 Score (F score)

How to compare precision/recall numbers?

	Precision(P)	Recall (R)	Average	F_1 Score
→ Algorithm 1	0.5	0.4	0.45	0.444 ←
→ Algorithm 2	0.7	0.1	0.4	0.175 ←
Algorithm 3	0.02	1.0	0.51	0.0392 ←

Predict y=1 all the time

Average: $\frac{P+R}{2}$

F_1 Score: $2\frac{PR}{P+R}$

P=0 or R=0 ⇒ F-score = 0.

P=1 and R=1 ⇒ F-score = 1

Using Large Datasets
Data for ML

Getting alot of data and training on a certain tye of learning algorithm, can be a very effective way to get a learning algorithm to do very good performance

Designing a high accuracy learning system

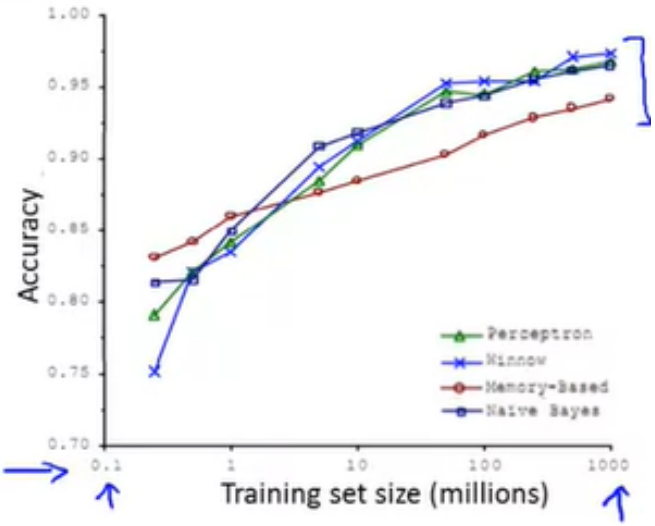
E.g. Classify between confusable words.

{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 data rationale

Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately.

Example: For breakfast I ate two eggs.

Counterexample: Predict housing price from only size (feet^2) and no other features.

Useful test: Given the input x, can a human expert confidently predict y?

just knowing one feature doesnt make you feel the correct value. be it a ML learning algo or a human expert

Large data rationale

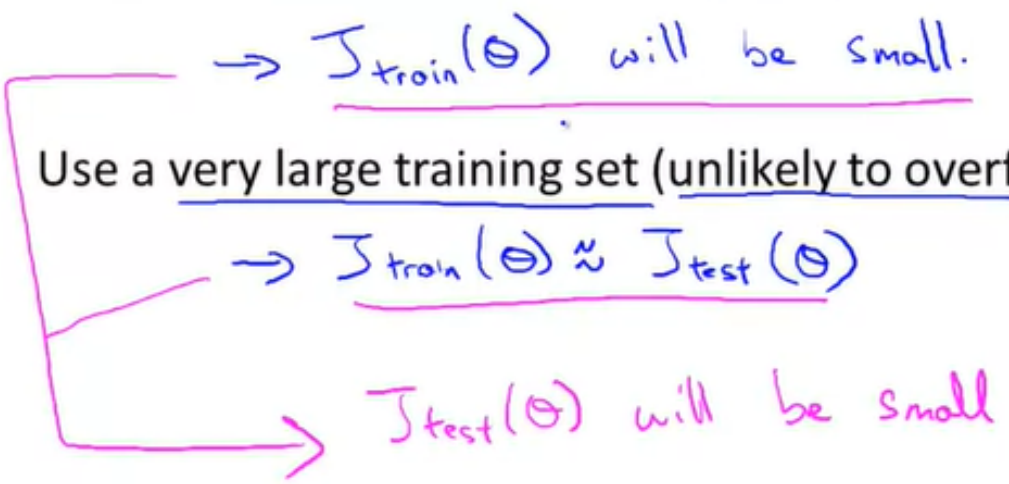
⇒ Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

→ $J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit) low variance ←

→ $J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ will be small



Having a large training set can help significantly improve a learning algorithm's performance. However, the large training set is **unlikely** to help when:

- [x] The features x do not contain enough information to predict y accurately (such as predicting a house's price from only its size), and we are using a simple learning algorithm such as logistic regression.

✓ **Correct**

training error will be close to test error - unlikely to overfit
large training set - low variance is considered
can a human expert look at the features x and confidently predict the value y ? one of the most important things to rem while solving the problem

	Actual Class: 1	Actual Class: 0
Predicted Class: 1	85	890
Predicted Class: 0	15	10

For reference:

- Accuracy = (true positives + true negatives) / (total examples)
- Precision = (true positives) / (true positives + false positives)
- Recall = (true positives) / (true positives + false negatives)
- F_1 score = (2 * precision * recall) / (precision + recall)

What is the classifier's precision (as a value from 0 to 1)?

Enter your answer in the box below. If necessary, provide at least two values after the decimal point.

0.08

✓ **Correct**

There are 85 true positives and 890 false positives, so precision is 85 / (85 + 890) = 0.087.

3. Suppose you have trained a logistic regression classifier which is outputting $h_\theta(x)$. **1 / 1 point**

Currently, you predict 1 if $h_\theta(x) \geq \text{threshold}$, and predict 0 if $h_\theta(x) < \text{threshold}$, where currently the threshold is set to 0.5.

Suppose you **increase** the threshold to 0.7. Which of the following are true? Check all that apply.

- [ ] The classifier is likely to have unchanged precision and recall, but higher accuracy.
- [ ] The classifier is likely to now have lower precision.
- [ ] The classifier is likely to have unchanged precision and recall, but lower accuracy.
- [x] The classifier is likely to now have lower recall.

✓ **Correct**

Increasing the threshold means more y = 0 predictions. This will increase the decrease of true positives and increase the number of false negatives, so recall will decrease.

- [ ] If your model is underfitting the training set, then obtaining more data is likely to help.
- [x] The "error analysis" process of manually examining the examples which your algorithm got wrong can help suggest what are good steps to take (e.g., developing new features) to improve your algorithm's performance.

✓ **Correct**
This process of error analysis is crucial in developing high performance learning systems, as the space of possible improvements to your system is very large, and it gives you direction about what to work on next.

- [ ] After training a logistic regression classifier, you **must** use 0.5 as your threshold for predicting whether an example is positive or negative.
- [ ] It is a good idea to spend a lot of time collecting a **large** amount of data before building your first version of a learning algorithm.
- [x] Using a **very large** training set makes it unlikely for model to overfit the training data.

✓ **Correct**
A sufficiently large training set will not be overfit, as the model cannot overfit some of the examples without doing poorly on the others.

The classifier achieves 99% accuracy because of the skewed classes in the data, not because it is overfitting the training set. Thus, it is likely to perform just as well on the cross validation set.

Week7

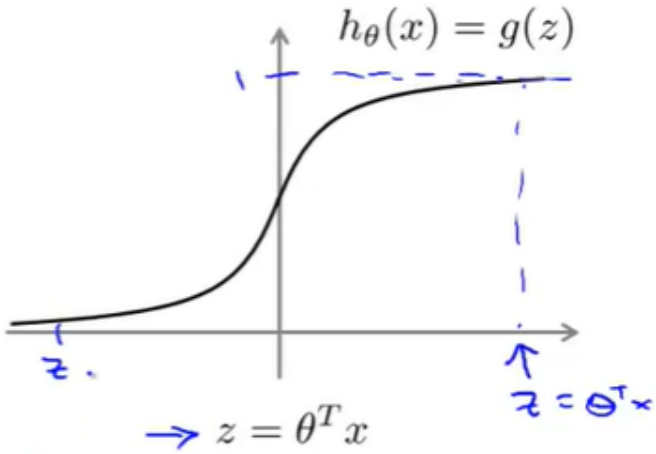
we have large margin classification, kernals, SVMs in practice

SVM - support vector machine
Optimization Objective

amount of data , algorithms used
very powerful algo - support vector machine - cleaner in terms of nonlinear functions

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 $h_\theta(x) \approx 0$, $\theta^T x \ll 0$

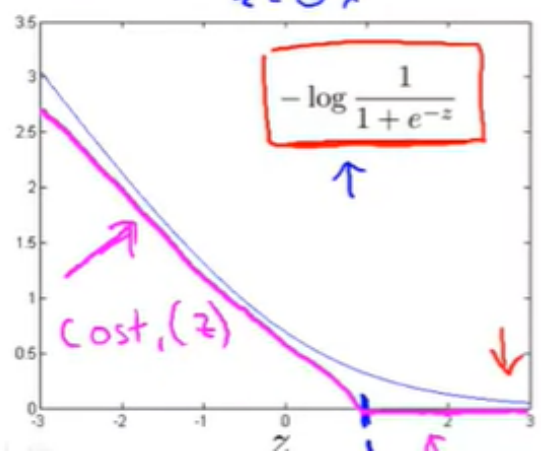
In case of y=0 ,
The purple curve is pretty close appox of the sigmoid curve shown
this purple curve gives a logistic regression and helps alot with the SVM

Alternative view of logistic regression

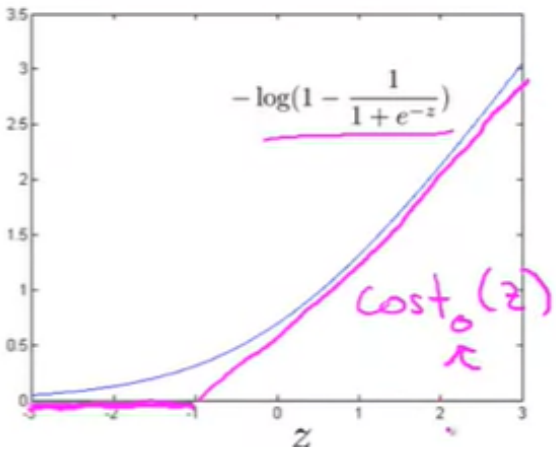
Cost of example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

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

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



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



for svm we paramertized sightly differently, at lambda

Support vector machine

Logistic regression:

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

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

Support vector machine:

$$\min_\theta \; C\sum_{i=1}^{m} y^{(i)}\,\text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\,\text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2}\sum_{i=0}^{n}\theta_j^2$$

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

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

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

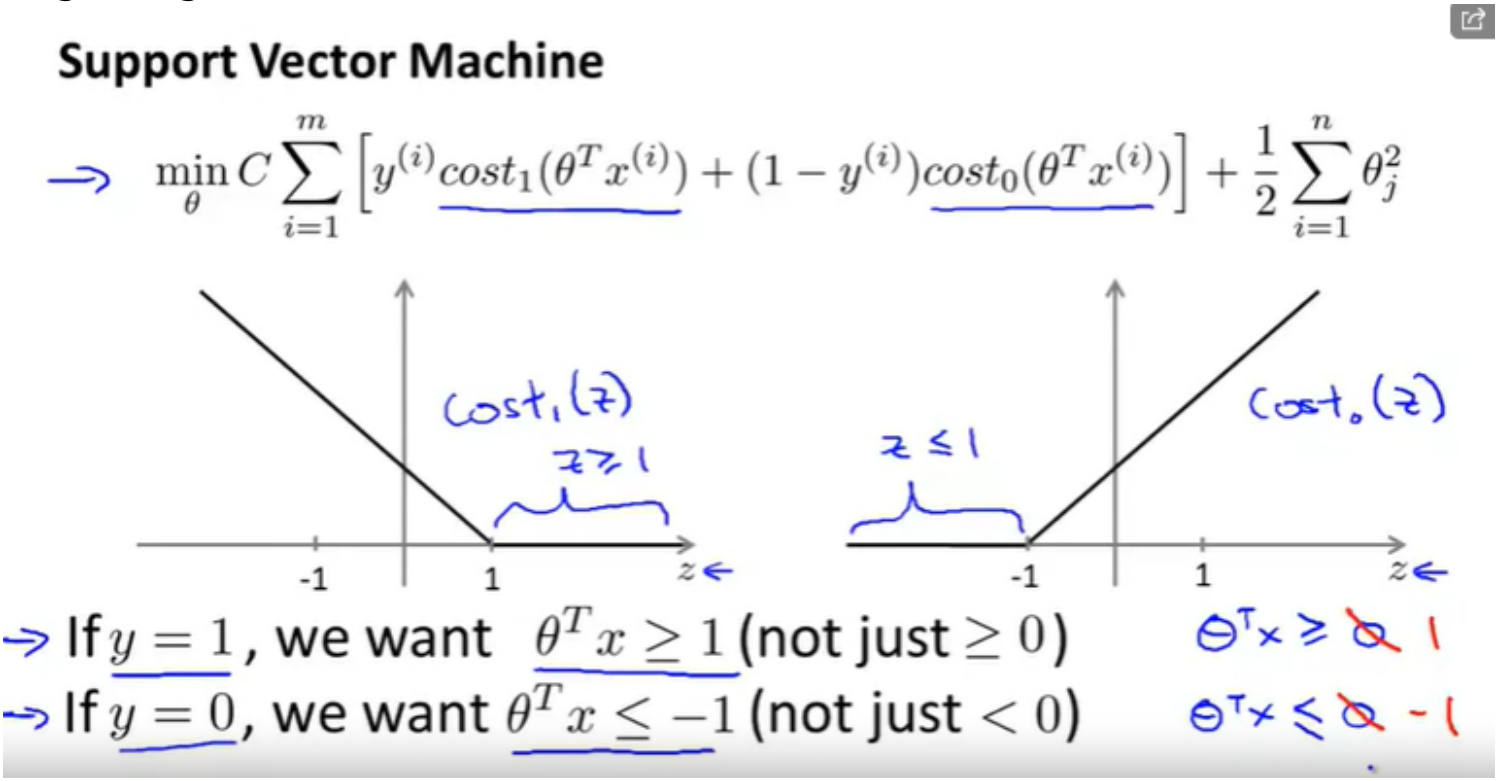
$CA + B$

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

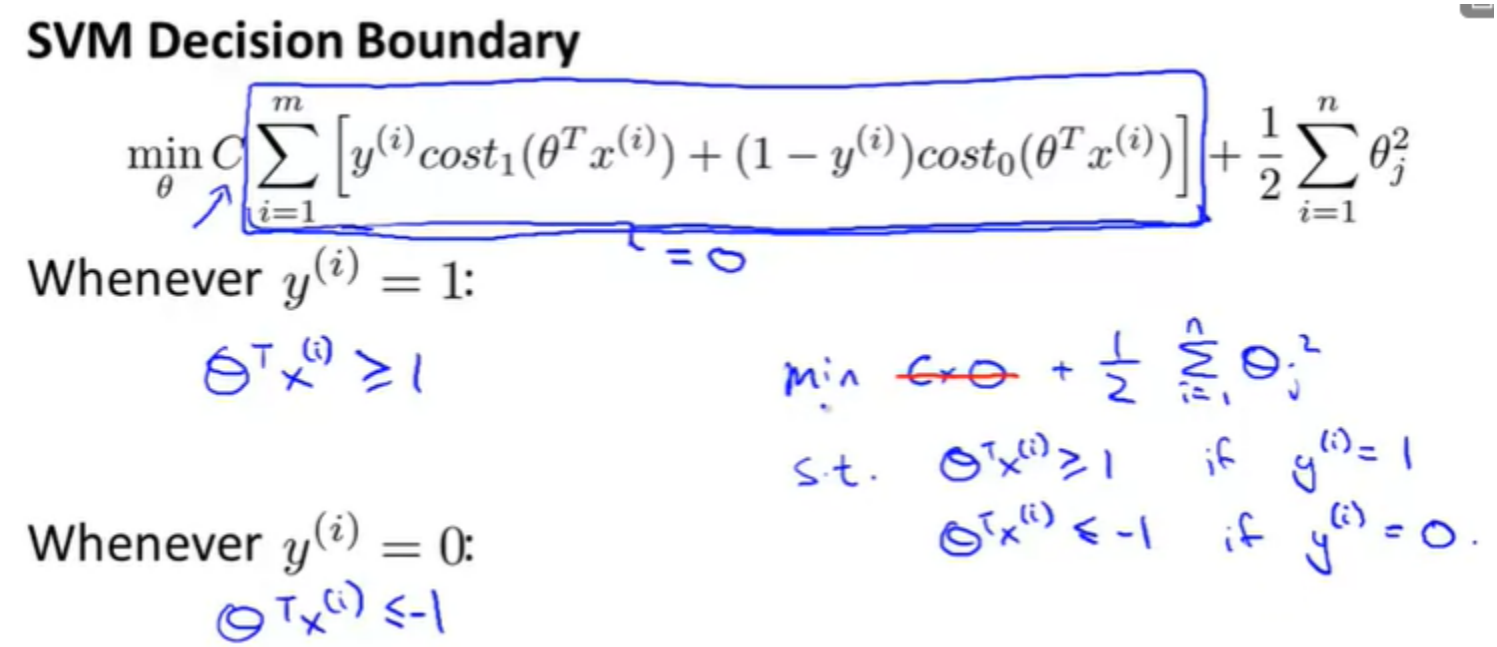
unlike logistic regression, the support vector machine doesn't output the probability is that what we have is we have this cost function, that we minimize to get the parameter's data, and what a support vector machine does is it just makes a prediction of y being equal to one or zero, directly. So the hypothesis will predict

one if theta transpose x is greater or equal to zero, and it will predict zero otherwise and so having learned the parameters theta, this is the form of the hypothesis for the support vector machine.

Large Margin Intuition

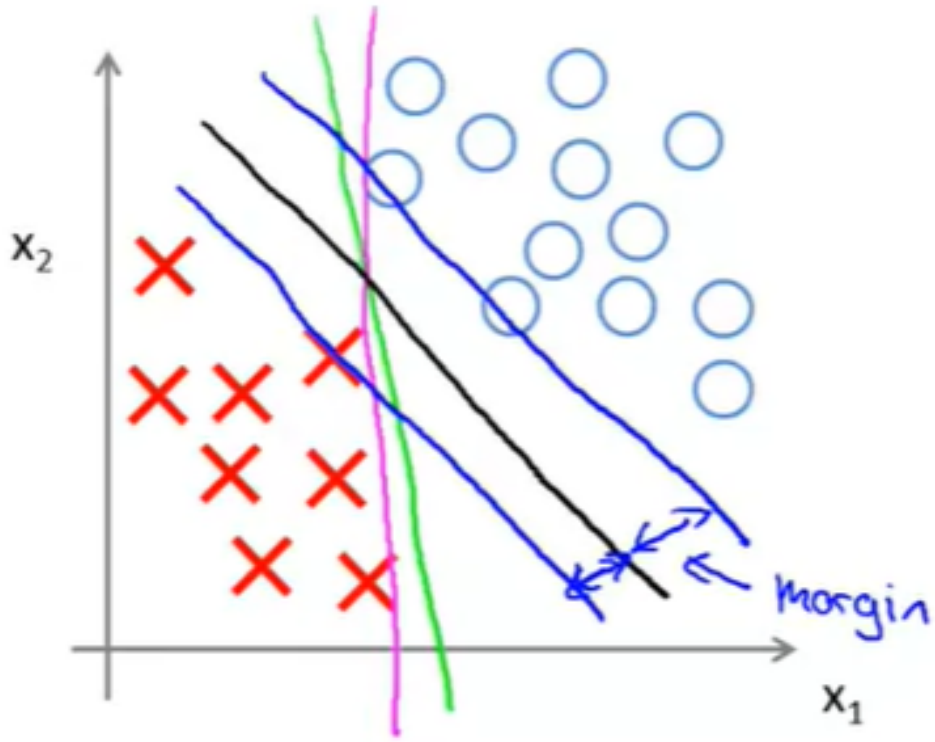


set C as very large value



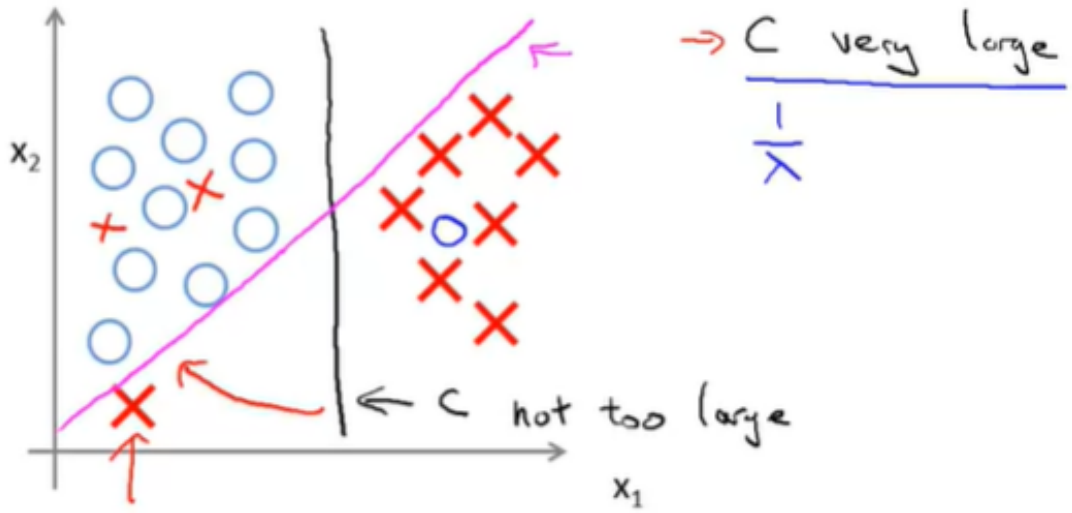
how does this large margin classifier end from the above formula

SVM Decision Boundary: Linearly separable case



Large margin classifier

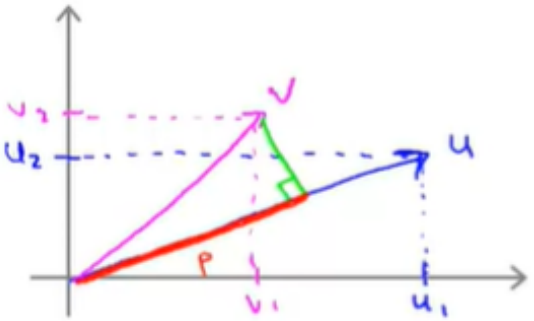
Large margin classifier in presence of outliers



I hope that gives some intuition about how this support vector machine functions as large margin classifier that tries to separate the data with a large margin, technically this picture of this view is true only when the parameter C is very large, which isa useful way to think about support vector machines

Mathematics behind large margin classification
inner product

Vector Inner Product



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

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

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

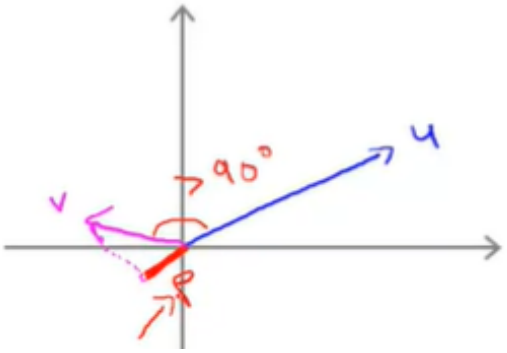
$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

p = length of projection of v onto u.

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

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

Signed



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

$p < 0$

$\omega = (\sqrt{\omega})^2$

SVM Decision Boundary

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

$= \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \frac{1}{2}\|\theta\|^2$

$= \|\theta\|$

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

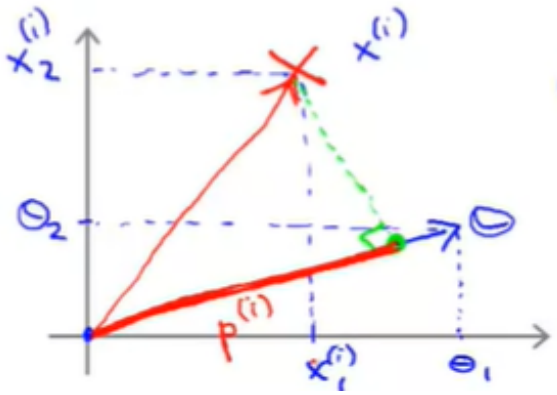
$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

Simplification: $\theta_0 = 0$. $\quad n = 2$

$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$ $\theta_0 = 0$

$\theta^T x^{(i)} = ?$

$u^T v$



$\theta^T x^{(i)} = p^{(i)} \cdot \|\theta\|$

$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$

Andrew N

|| theta || is norm of theta

SVM Decision Boundary

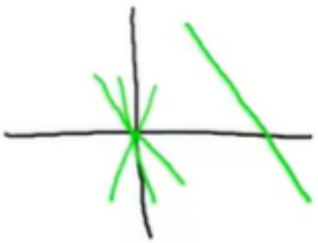
$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$$

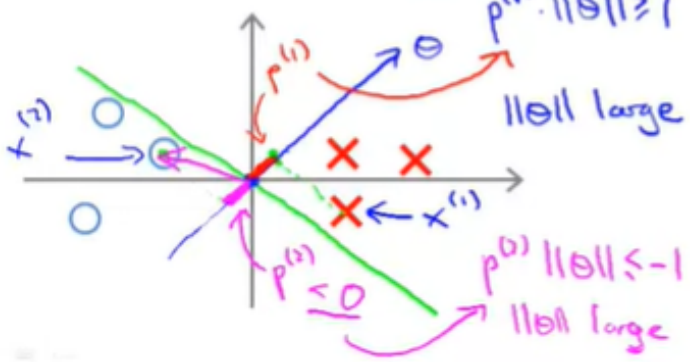
s.t. $p^{(i)} \cdot \|\theta\| \geq 1$ if $y^{(i)} = 1$

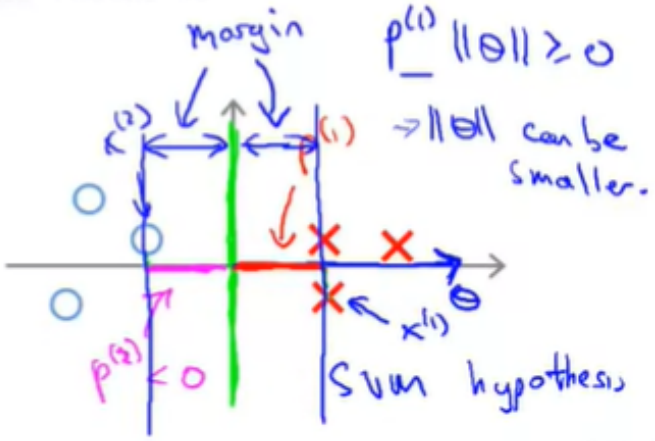
$p^{(i)} \cdot \|\theta\| \leq -1$ if $y^{(i)} = 1$

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

Simplification: $\theta_0 = 0$



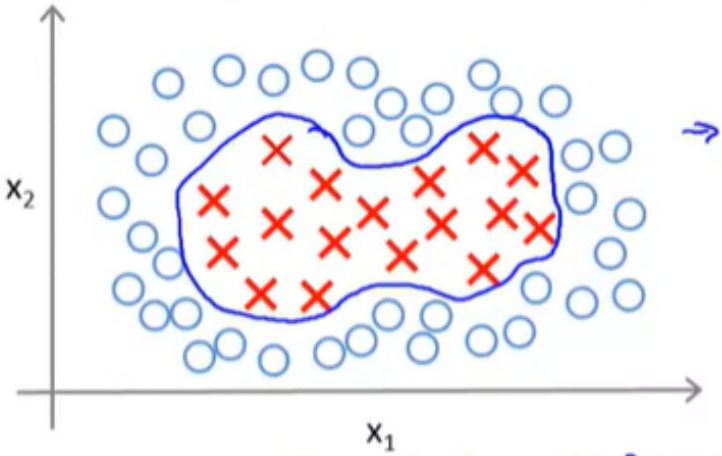




Kernel I

adapting the SVMs in order to develop complex nonlinear classifiers
main technique is use kernels

Non-linear Decision Boundary



Predict $y = 1$ if

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

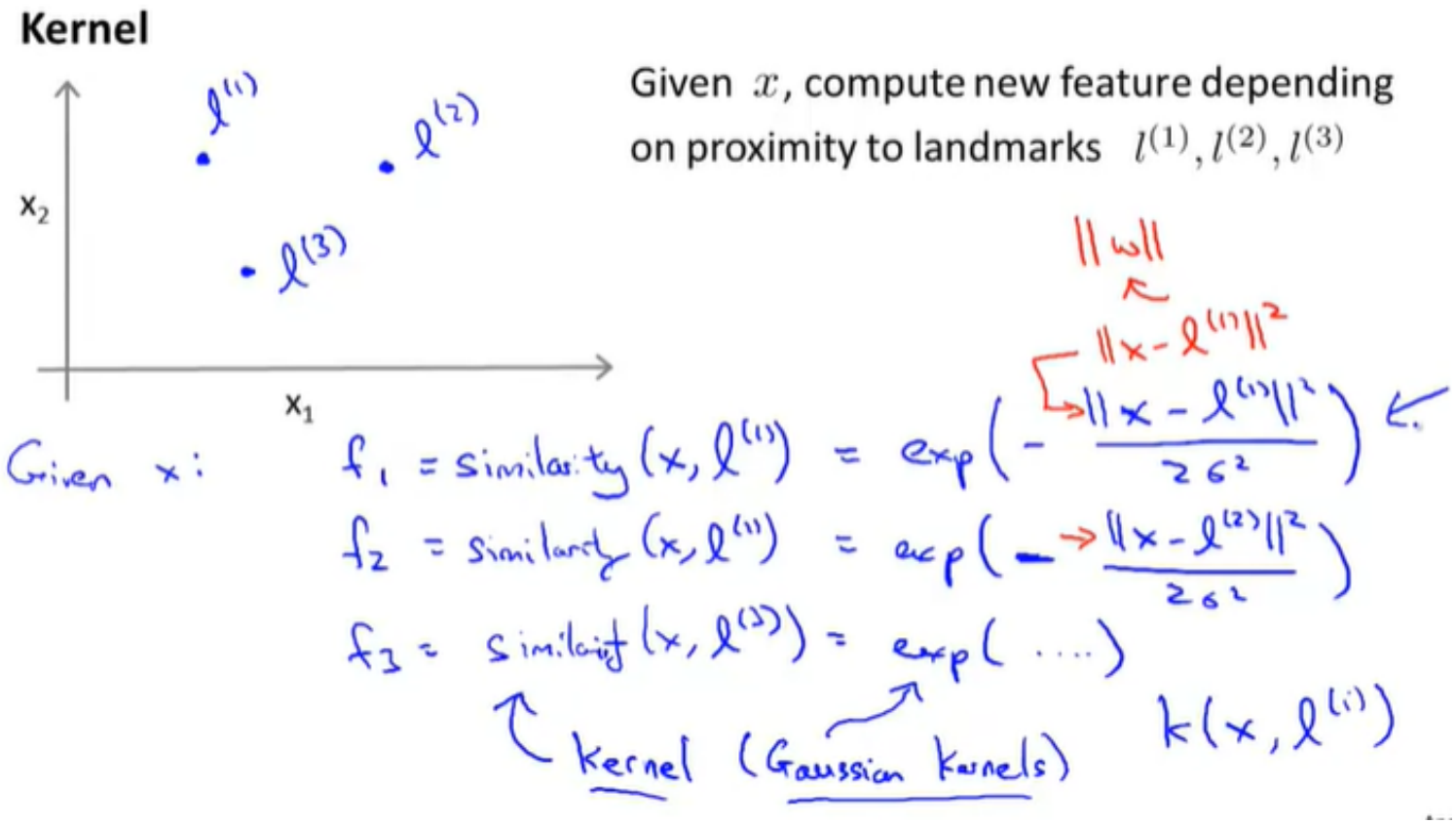
$h_\theta(x) = 1$ if $\theta_0 + \theta_1 x_1 + \cdots \geq 0$; 0 otherwise.

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

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

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

Define new examples as follows :
gaussian kernel formula



l- landmark
the question is, is there a different choice of features or is there better sort of features than this high order polynomials because you know it's not clear that this high order polynomial is what we want, and what we talked about computer vision talk about when the input is an image with lots of pixels.
We also saw how using high order polynomials becomes very computationally expensive because there are a lot of these higher order polynomial terms.
So, is there a different or a better choice of the features that we can use to plug into this sort of hypothesis form

So what these features do is they measure how similar X is from one of your landmarks and the feature f is going to be close to one when X is close to your landmark and is going to be 0 or close to zero when X is far from your landmark.
Each of these landmarks, defines a new feature f1, f2 and f3. That is, given the the training example X, we can now compute three new features: f1, f2, and f3, given, you know, the three landmarks that I wrote just now.
But first, let's look at this exponentiation function, let's look at this similarity function and plot in some figures and just, you know, understand better what this really looks like.

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

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

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

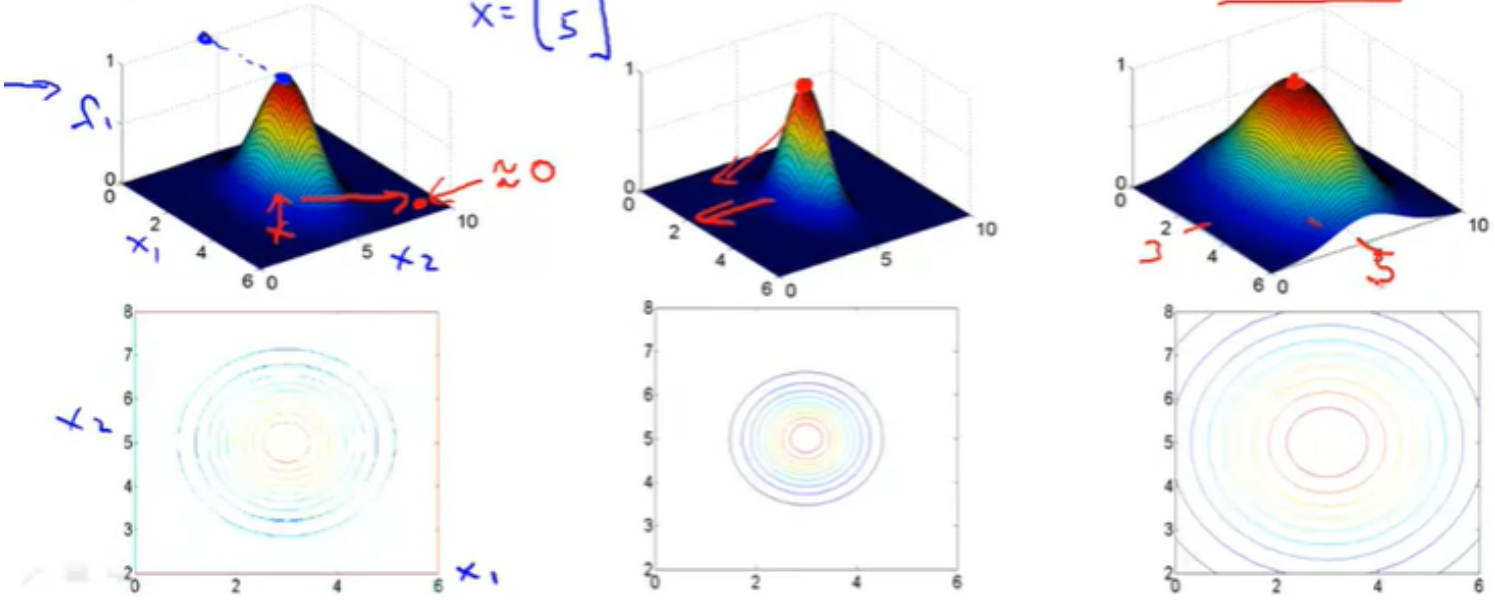
$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

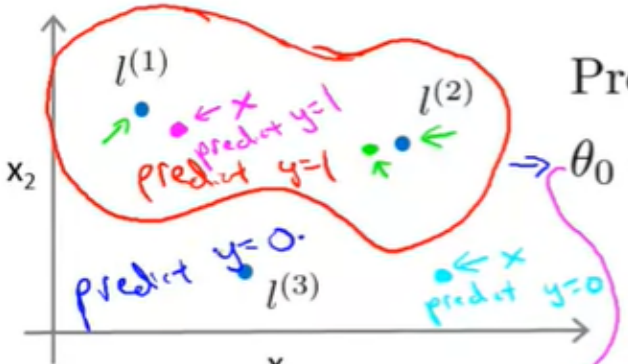
Now the other was due on this slide is show the effects of varying this parameter sigma squared. So, sigma squared is the parameter of the Gaussian kernel and as you vary it, you get slightly different effects.
Let's set sigma squared to be equal to 0.5 and see what we get. We set sigma square to 0.5, what you find is that the kernel looks similar, except for the width of the bump becomes narrower.
The contours shrink a bit too. So if sigma squared equals to 0.5 then as you start from X equals 3 5 and as you move away,then the feature f1 falls to zero much more rapidly and conversely.And if sigma squared is large, then as you move away from l1, the value of the feature falls away much more slowly.

Example:

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

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





Predict "1" when

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

$\theta_0 = -0.5$, $\theta_1 = 1$, $\theta_2 = 1$, $\theta_3 = 0$

$f_1 \approx 1$, $f_2 \approx 0$, $f_3 \approx 0$.

$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0$

$= -0.5 + 1 = 0.5 \geq 0$

$f_1, f_2, f_3 \approx 0$

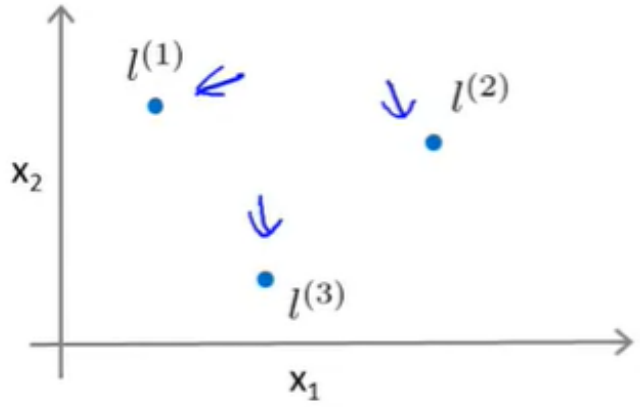
$\theta_0 + \theta_1 f_1 + \dots \approx -0.5 < 0$

We can learn pretty complex non-linear decision boundary, like what I just drew where we predict positive when we're close to either one of the two landmarks. And we predict negative when we're very far away from any of the landmarks. And so this is part of the idea of kernels of and how we use them with the support vector machine, which is that we define these extra features using landmarks and similarity functions to learn more complex nonlinear classifiers.

Kernel II

yes how do we get landmarks

Choosing the landmarks

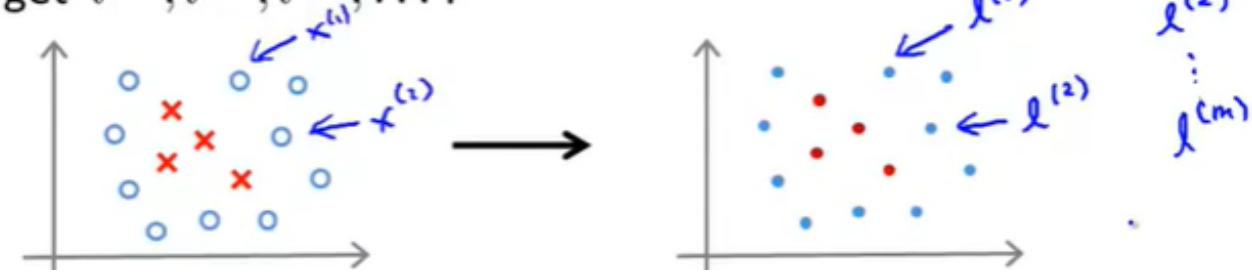


Given x:

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

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



similarities

SVM with Kernels

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

choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x:

$$f_1 = \text{similarity}(x, l^{(1)})$$
$$f_2 = \text{similarity}(x, l^{(2)})$$
$$\ldots$$

$x^{(i)}$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \quad f_0 = 1$$

For training example $(x^{(i)}, y^{(i)})$:

$x^{(i)} \rightarrow$ $f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$, $f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$, $\ldots$, $f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = \exp(-\frac{0}{2\sigma^2}) = 1$, $\ldots$, $f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

$x^{(i)} \in \mathbb{R}^{n+1}$ (or $\mathbb{R}^n$)

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \quad f_0^{(i)} = 1$$

this is the idea here which is that we're gonna take the examples and for every training example that we have, we are just going to call it. We're just going to put landmarks as exactly the same locations as the training examples

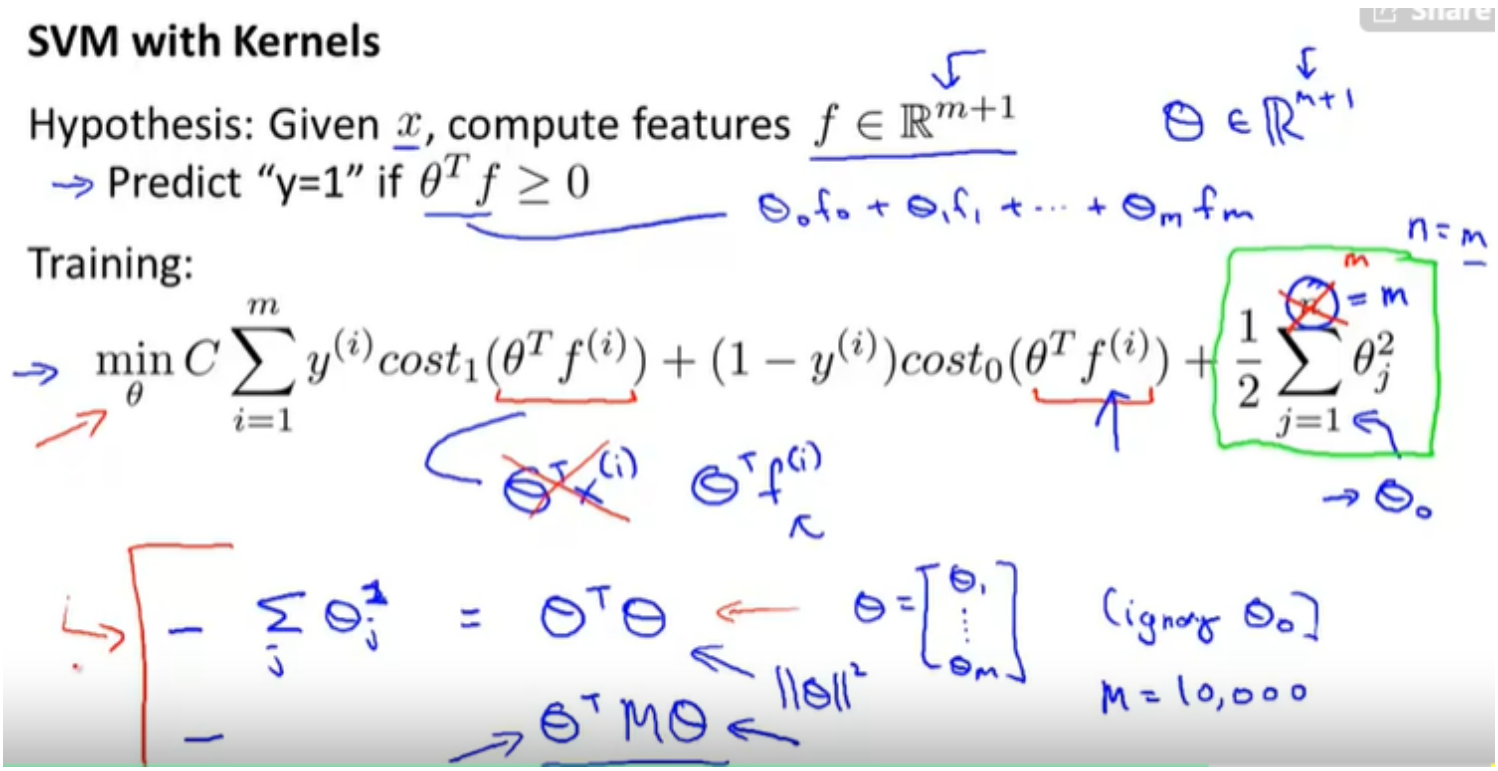
When you are given example x, and in this example x can be something in the training set, it can be something in the cross validation set, or it can be something in the test set. Given an example x we are going to compute, you know, these features as so f1, f2, and so on. Where l1 is actually equal to x1 and so on. And these then give me a feature vector. So let me write f as the feature vector. I'm going to take these f1, f2 and so on, and just group them into feature vector

Depending on whether you can set terms, is either R(n) or R(n) plus 1. We can now instead represent my
training example using this feature vector f. I am
going to write this f superscript i. Which is going to be taking all of these things and stacking them into a vector.So, f1(i) down to fm(i) and if you want and well, usually we'll also add this f0(i), where f0(i) is equal to 1. And so this vector here gives me my new feature vector with which to represent my training example. So given these kernels and similarity functions, here's how we use a simple vector machine

So that's how you make a prediction if you already have a setting for the parameter's theta. How do you get the parameter's theta? Well you do that using the SVM learning algorithm, and specifically what you do is you would solve this minimization problem. You've minimized the parameter's theta of C times this cost function which we had before.
Only now, instead of looking there instead of making predictions using theta transpose x(i) using our original features, x(i). Instead we've taken the features x(i) and replace them with a new features

And what most support vector machine implementations do is actually replace this theta transpose theta, will instead, theta transpose times some matrix inside, that depends on the kernel you use, times theta. And so this gives us a slightly different distance metric. We'll use a slightly different measure instead of minimizing exactly the norm of theta squared means that minimize something slightly similar to it. That's like a rescale version of the parameter vector theta that depends on the kernel. But this is kind of a mathematical detail.

That allows the support vector machine software to run much more efficiently. And the reason the support vector machine does this is with this modification. It allows it to scale to much bigger training sets. Because for example, if you have a training set with 10,000 training examples.



bias and variance trade off

When using an SVM, one of the things you need to choose is the parameter C which was in the optimization objective, and you recall that C played a role similar to 1 over lambda, where lambda was the regularization parameter we had for logistic regression.So, if you have a large value of C, this corresponds to what we have back in logistic regression, of a small value of lambda meaning of not using much regularization. if you do that, you tend to have a hypothesis with lower bias and higher variance.

Whereas if you use a smaller of C then this corresponds to when we are using logistic regression with a large value of lambda and that corresponds to a hypothesis with higher bias and lower variance. And so, hypothesis with large C has a higher variance, and is more prone to overfitting, whereas hypothesis with small C has higher bias and is thus more prone to underfitting.

SVM parameters:

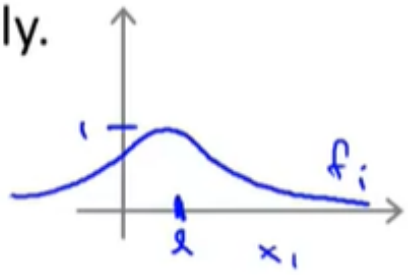
C ($= \frac{1}{\lambda}$). → Large C: Lower bias, high variance. (small λ)

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

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

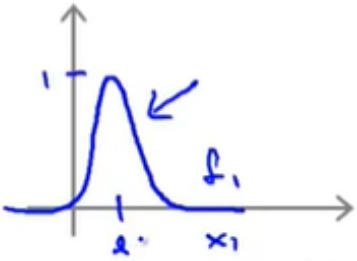
→ Higher bias, lower variance.

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



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



Using an SVM

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

Andrew

Kernel (similarity) functions:

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

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

$\|x - l\|^2$ $\quad v = x - l$ $\quad x \in \mathbb{R}^n$

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

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

1000 feet² $\quad$ 1-5 bedrooms

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:

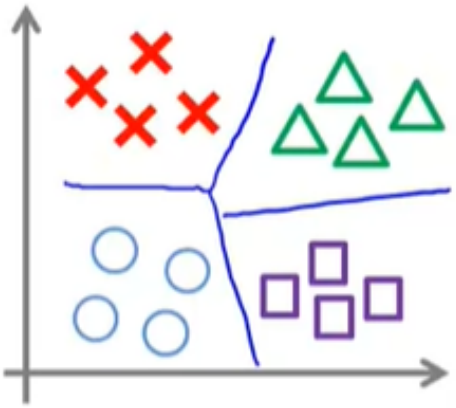
- Polynomial kernel: $k(x, l) = (x^T l)^2$, $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$, $(x^T l + \text{constant})^{\text{degree}}$
- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

$sim(x, l)$

Suppose you are trying to decide among a few different choices of kernel and are also choosing parameters such as C, σ^2, etc. How should you make the choice?

- ○ Choose whatever performs best on the training data.
- ◉ Choose whatever performs best on the cross-validation data.
- ○ Choose whatever performs best on the test data.
- ○ Choose whatever gives the largest SVM margin.

Multi-class classification



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

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

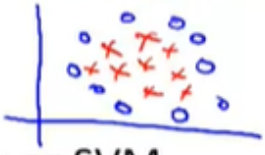
Otherwise, use one-vs.-all method. (Train K SVMs, one to distinguish $y = i$ from the rest, for $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

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

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.

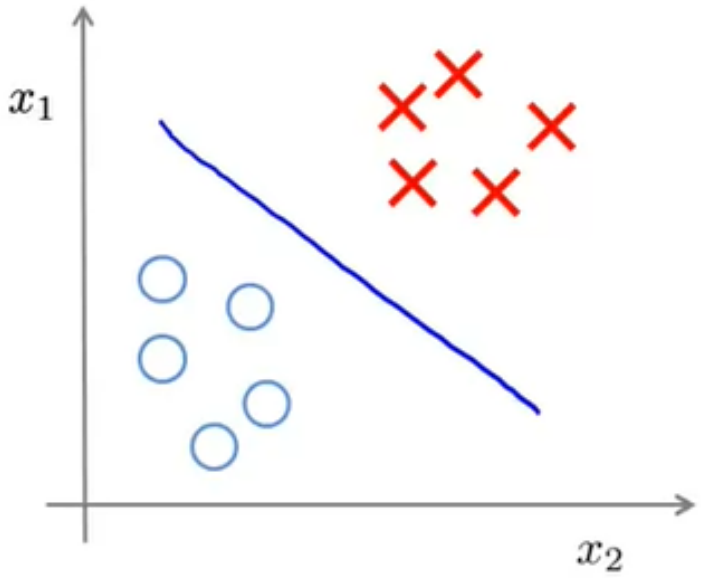
Week8

Unsupervised Learning : Introduction

Clustering

learn from unlabeled data - pretty exciting

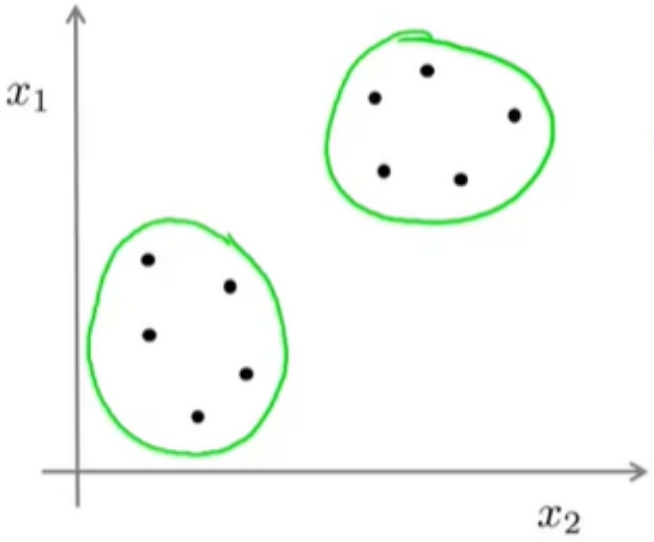
Supervised learning



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

clustering

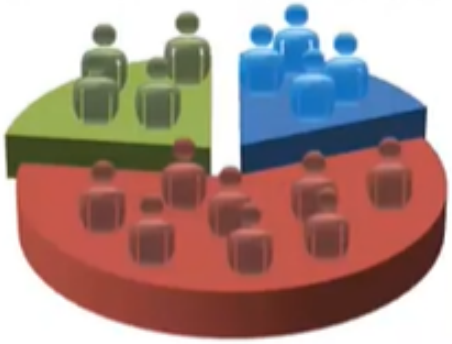
Unsupervised learning



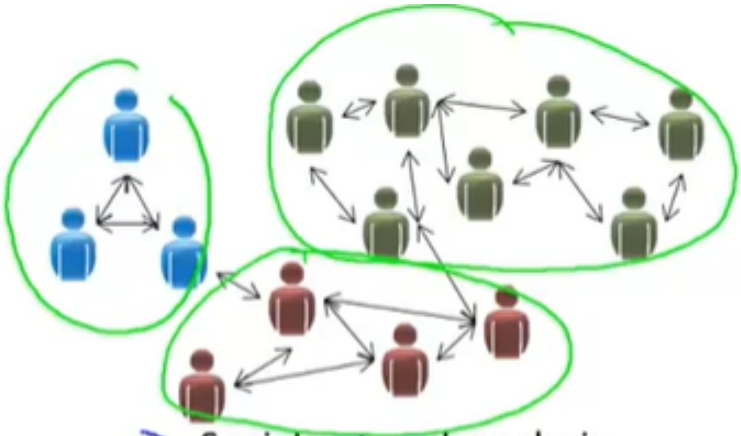
Clustering algorithm

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

Applications of clustering



→ Market segmentation



→ Social network analysis

Organize computing clusters

→ Astronomical data analysis

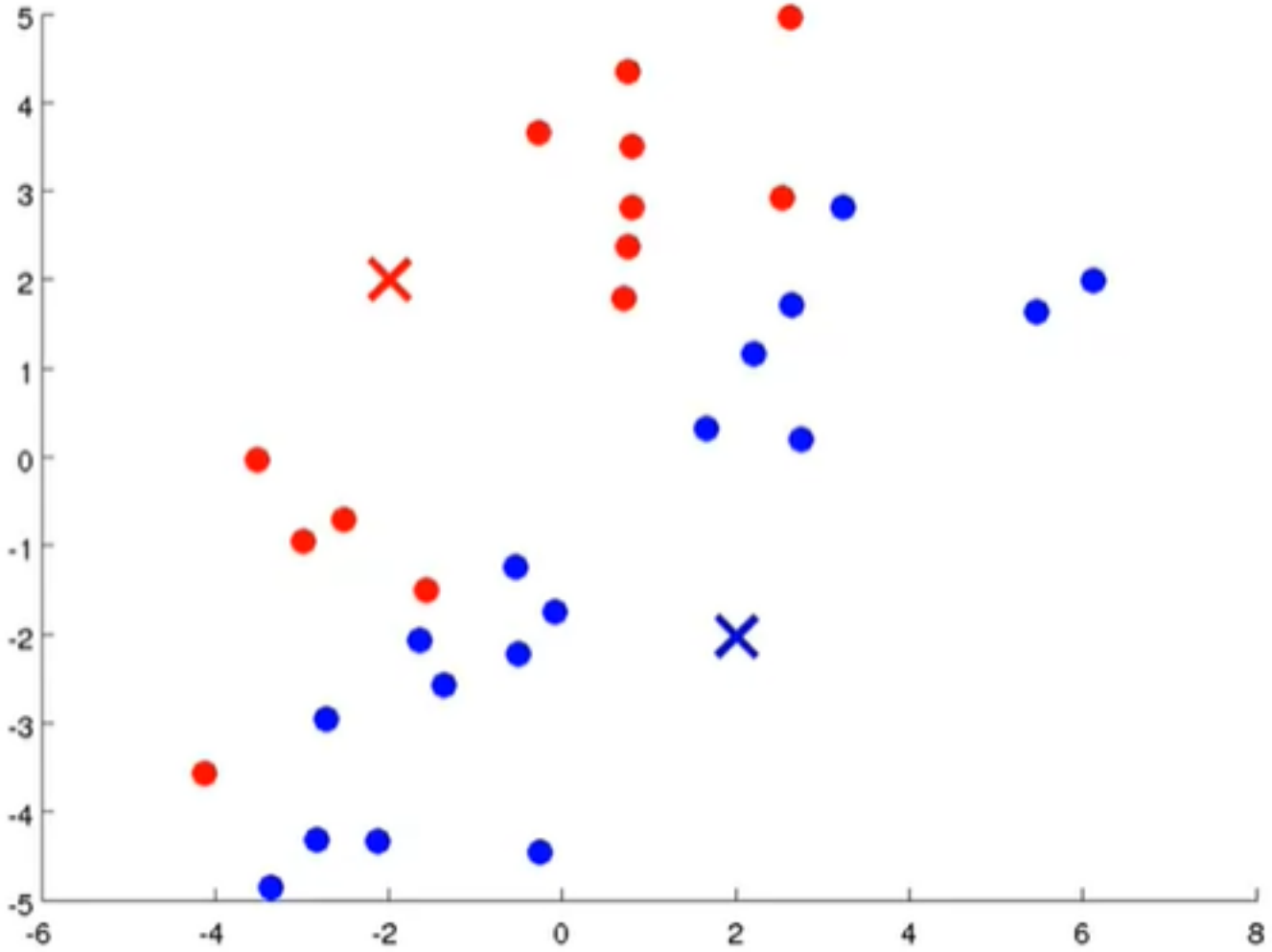
K means algorithm

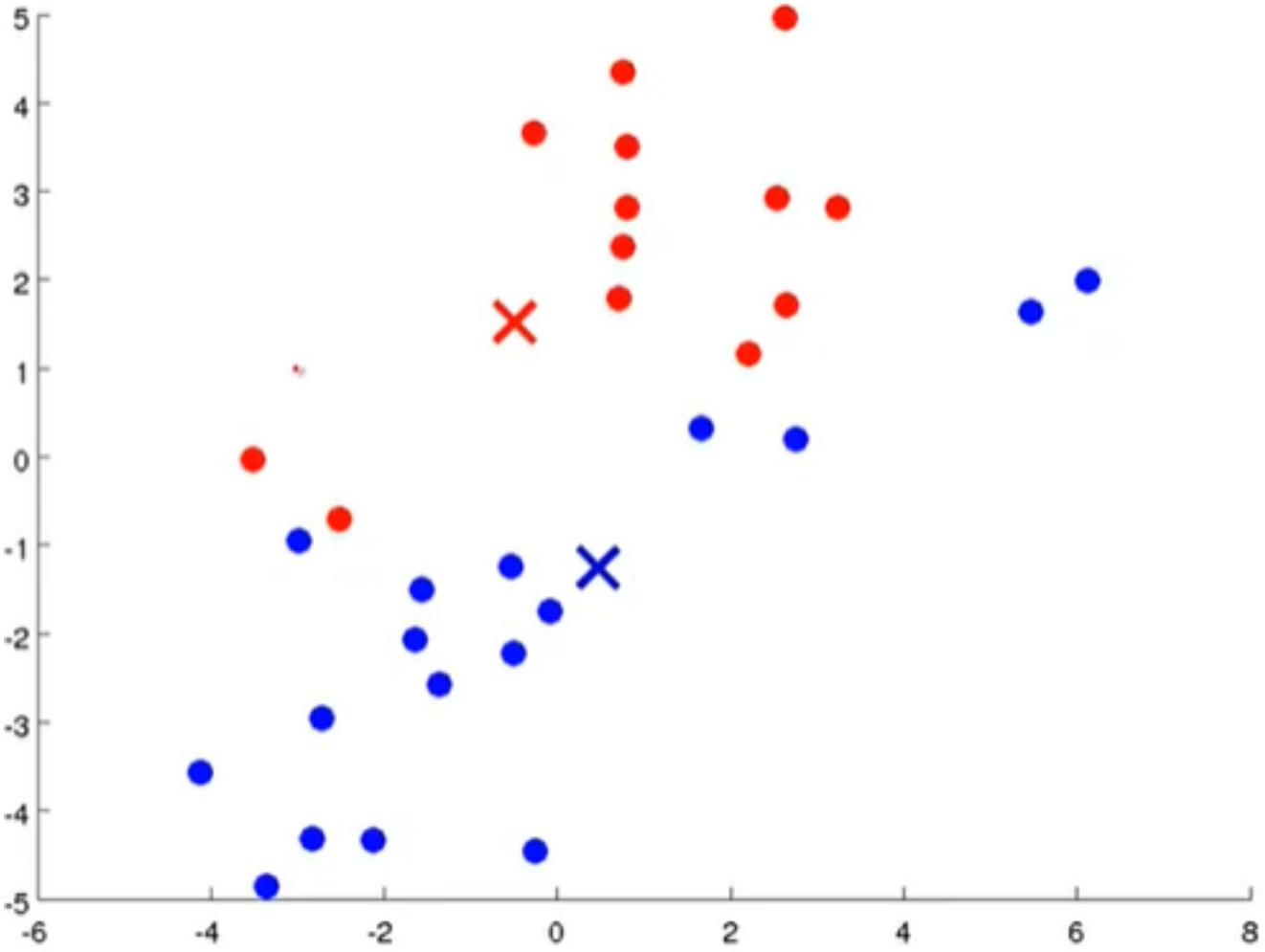
If I run the K Means clustering algorithm, here is what
I'm going to do. The first step is to randomly initialize two points, called the cluster centroids.
So, these two crosses here, these are called the Cluster Centroidsand I have two of

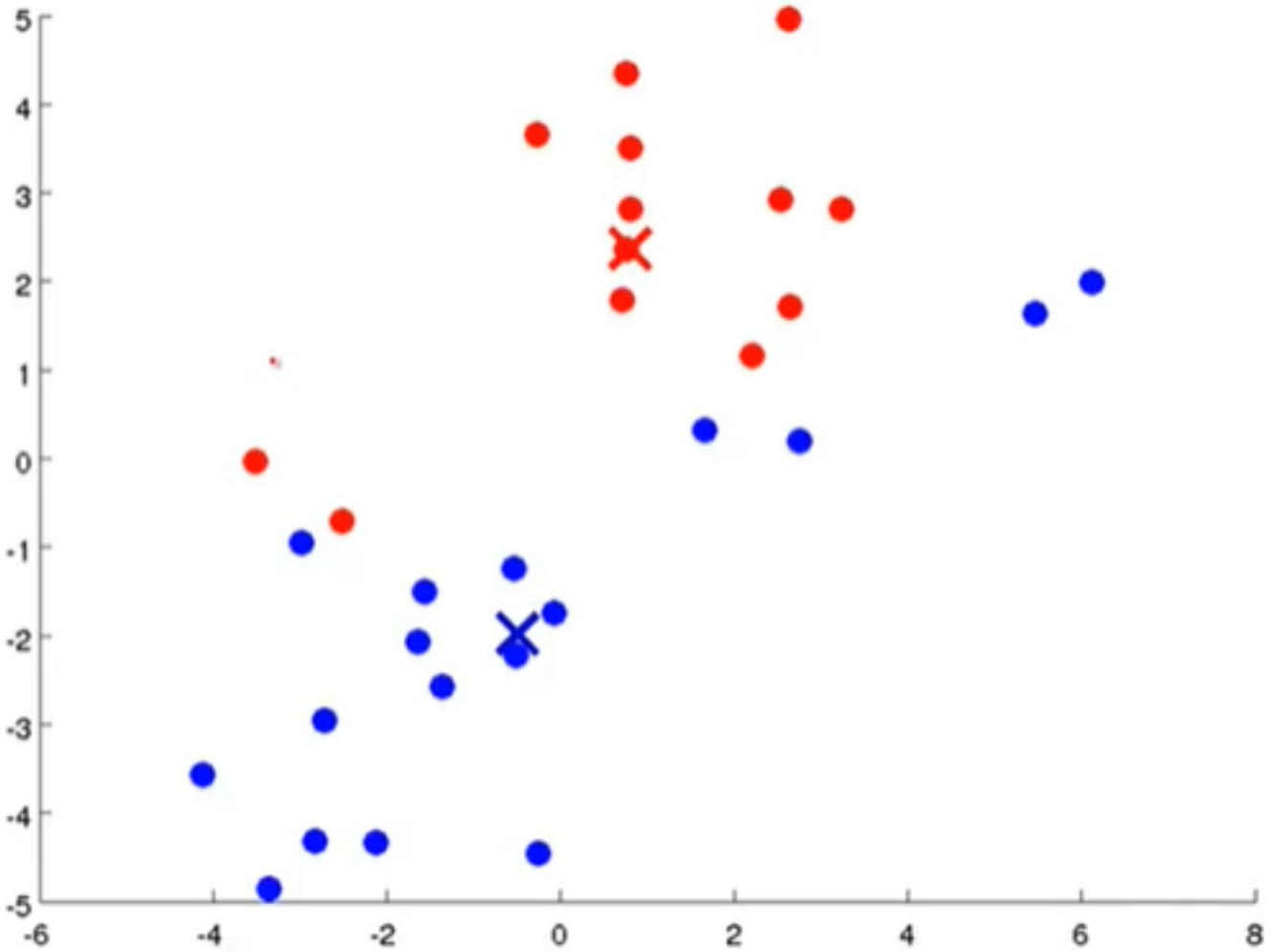
them
because I want to group my data into two clusters.
K Means is an iterative algorithm and it does two things.
First is a cluster assignment step, and second is a move centroid step. So, let me tell you what those things mean.

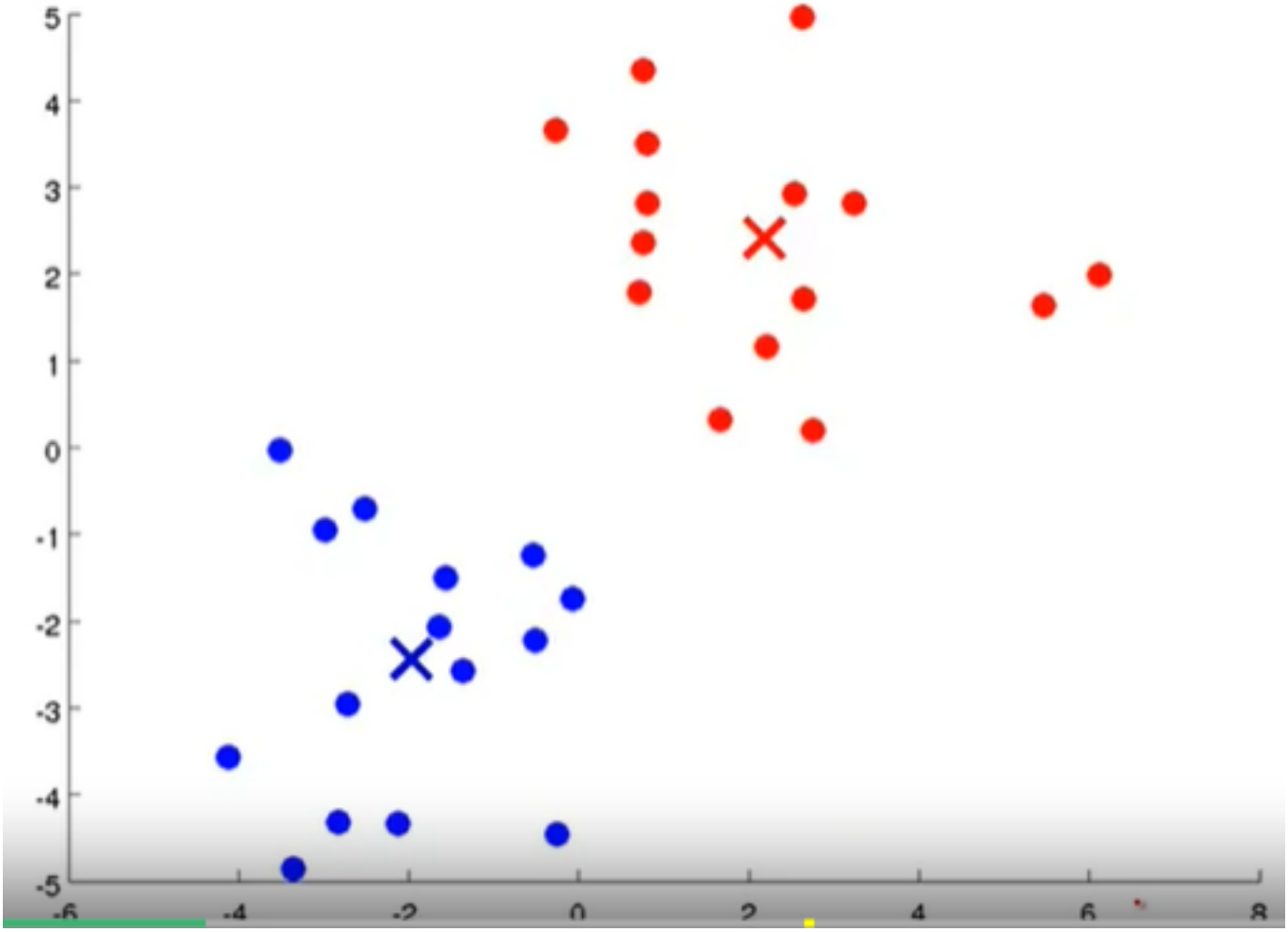
The first of the two steps in the loop of K means, is this cluster assignment step. What that means is that, it's going through each of the examples, each of these green dots shown here and depending on whether it's closer to the red cluster centroid or the blue cluster centroid, it is going to assign each of the data points to one of the two cluster centroids.

Specifically, what I mean by that, is to go through your data set and color each of the points either red or blue, depending on whether it is closer to the red cluster centroid or the blue cluster centroid, and I've done that in this diagram here.




5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8



if you keep running additional iterations of K means from here the cluster centroids will not change any further and the colours of the points will not change any further. And so, this is the, at this point, K means has converged and it's done a pretty good job findingthe two clusters in this data

if you keep running additional iterations of K means from here the cluster centroids will not change any further and the colours of the points will not change any further. And so, this is the, at this point, K means has converged and it's done a pretty good job findingthe two clusters in this data

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)

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 $x^{(i)}$

$\min_k \|x^{(i)} - \mu_k\|^2$

Move centroid

for k = 1 to K

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

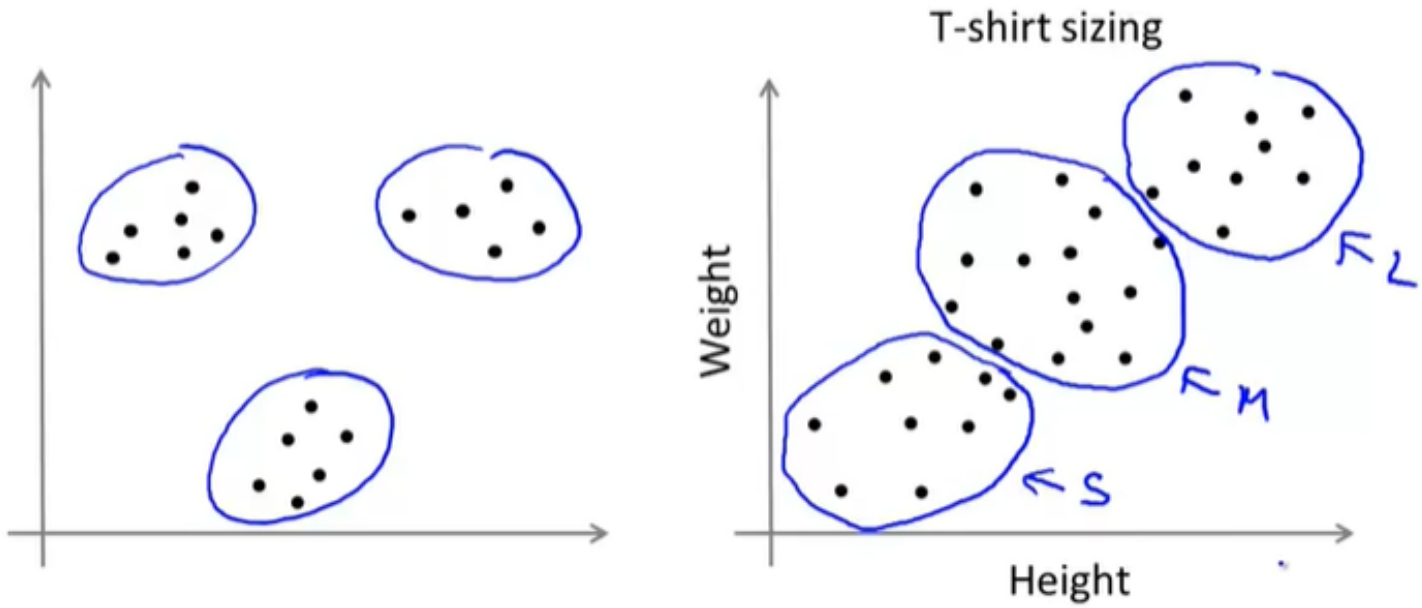
$x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)} \rightarrow c^{(1)}=2,\ c^{(5)}=2,\ c^{(6)}=2,\ c^{(10)}=2$

$\mu_2 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right] \in \mathbb{R}^n$

}

K-means for non-separated clusters

S, M, L



Optimization techniques

K-means optimization objective
K - total no of clusters
k = {1,2,3,....K}

The cost function that k-means is minimizing is a function J of all of these parameters, c1 through cm and mu 1 through mu K. That k-means is varying as the algorithm runs. And the optimization objective is shown to the right, is the average of 1 over m of sum from i equals 1 through m of this term here.

Trying to find c and mu to try to minimize this cost function J. This cost function is sometimes also called the distortion cost function, or the distortion of the k-means algorithm.

K-means optimization objective

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

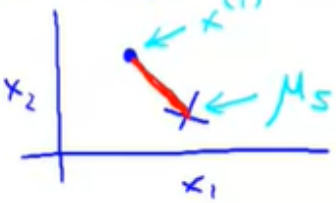
→ μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

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

Optimization objective:

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

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



Cluster assignment step - assign each point to the closest centroid
the cluster assignment step does is it doesn't change the cluster centroids, but what it's doing is this is exactly picking the values of c1, c2, up to cm. That minimizes the cost function, or the distortion function J.
The second step : K means was the move centroid step.
taking the two sets of variables and partitioning them into two halves right here. First the c sets of variables and then you have the mu sets of variables. And what it does is it first minimizes J with respect to the variable c and then it minimizes J with respect to the variables mu and then it keeps on.
We can use that to debug k-means and help make sure that k-means is converging and is running properly.

K-means algorithm

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

Repeat {

Cluster assignment step

Minimize $J(\ldots)$ wrt $c^{(1)}, c^{(2)}, \ldots, c^{(m)}$ ←

(holding $\mu_1, \ldots, \mu_K$ fixed)

for i = 1 to m

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

Move centroid

for k = 1 to K

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

}

minimize $J(\ldots)$ wrt $\mu_1, \ldots, \mu_K$

Random Initialization

how to make K-means avoid local optima as well.
m-training examples

Random initialization

Should have $K < m$

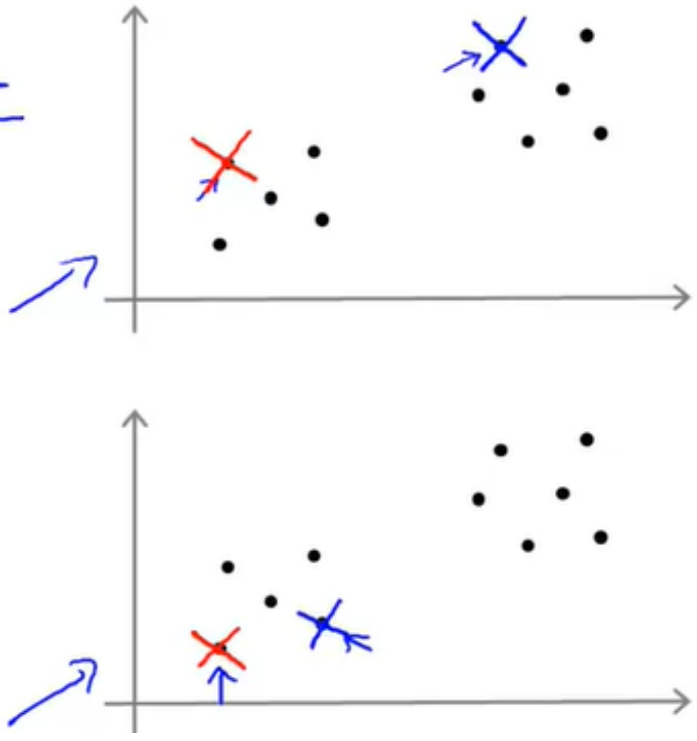
$K=2$

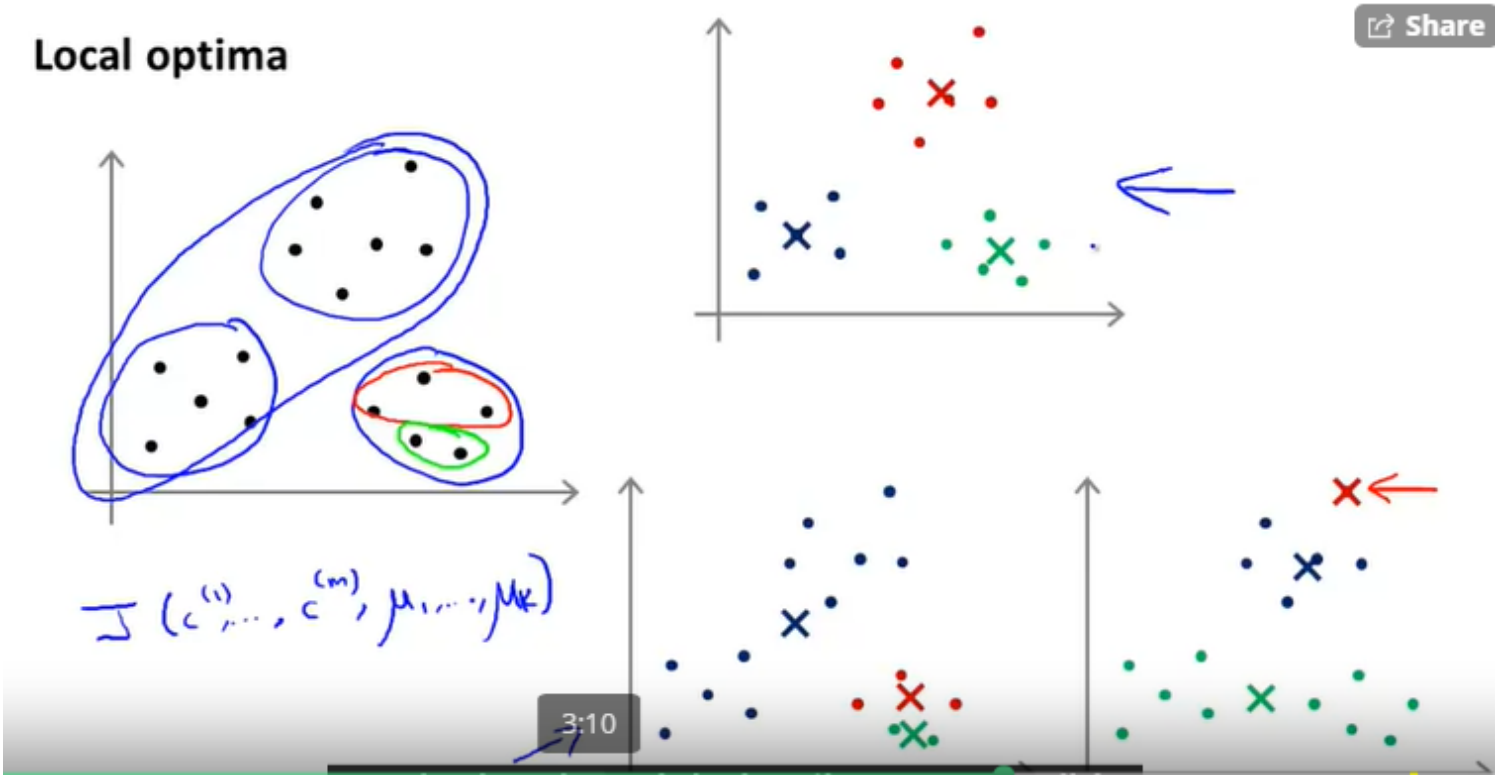
Randomly pick K training examples.

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

$\mu_1 = x^{(i)}$

$\mu_2 = x^{(j)}$





instead of just initializing K-means once and hopping that that works, what we can do is, initialize K-means lots of times and run K-means lots of times, and use that to try to make sure we get as good a solution, as good a local or global optima as possible.

You will have a hundred different ways of clustering the data and then finally what you do is all of these hundred ways you have found of clustering the data, just pick one, that gives us the lowest cost.
That gives us the lowest distortion. And it turns out that if you are running K-means with a fairly small number of clusters , so you know if the number of clusters is anywhere from two up to maybe 10 - then doing multiple random initializations can often, can sometimes make sure that you find a better local optima. Make sure you find the better clustering data. But if K is very large, so, if K is much greater than 10, certain if K were, you know, if you were trying to find hundreds of clusters, then,having multiple random initializations is less likely to make a huge difference and there is a much higher chance that your first random initialization will give you a pretty decent solution already

Random initialization

For i = 1 to 100 {

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

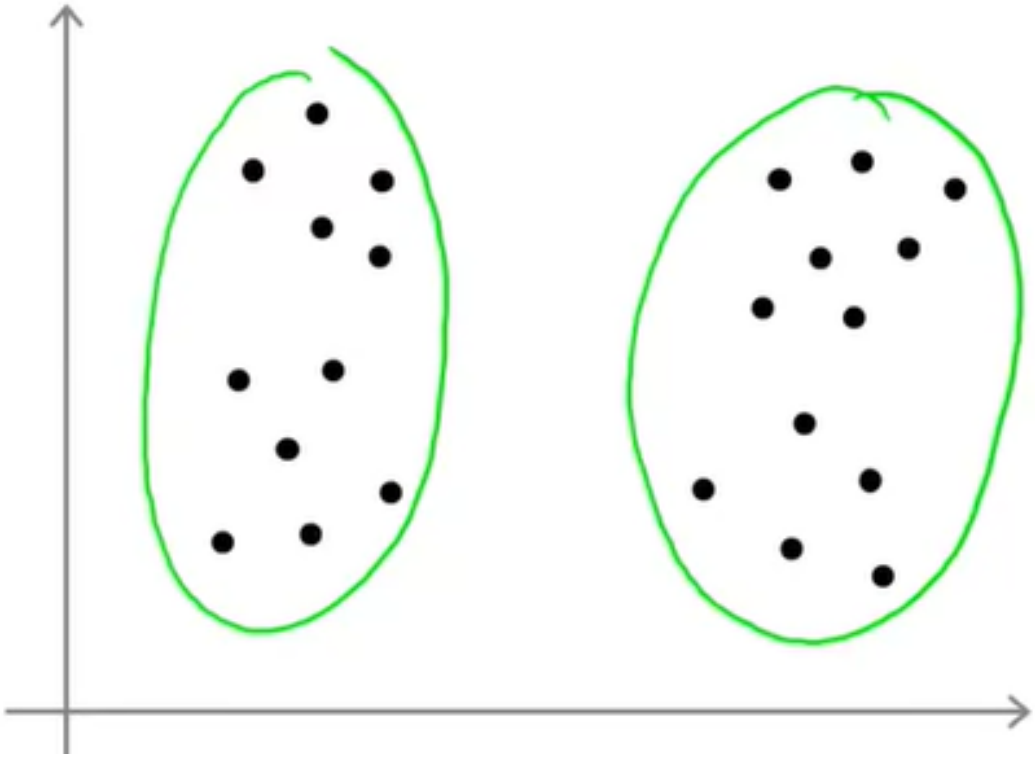
50 - 1000

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

K = 2 - 10

Choosing the number of clusters

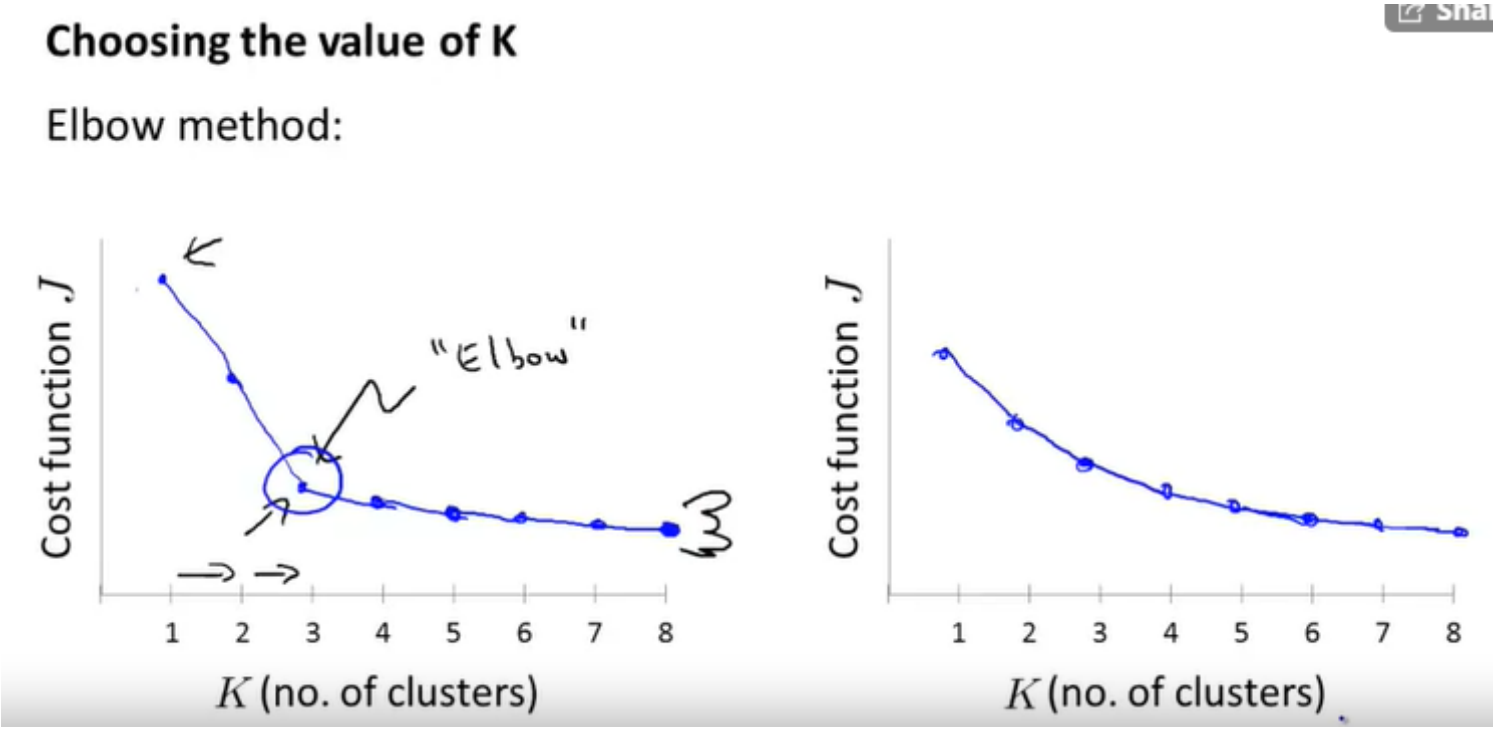
What is the right value of K?



the Elbow Method, what we're going to do is vary K, which is the total number of clusters. So, we're going to run K-means with one cluster, that means really, everything gets grouped into a single cluster and compute the cost function or compute the distortion J and plot that here. And then we're going to run K means with two clusters, maybe with multiple random initial agents, maybe

not. But then, you know, with two clusters we should get, hopefully, a smaller distortion, and so plot that there.
And then run K-means with three clusters, hopefully, you get even smaller distortion and plot that there.
I'm gonna run K-means with four, five and so on.

if you actually do this in a practice, you know, if your plot looks like the one on the left and that's great.
It gives you a clear answer, but just as often, you end up with a plot that looks like the one on the right and is not clear where the ready location of the elbow is. It makes it harder to choose a number of clusters using this method. So maybe the quick summary of the Elbow Method is that is worth the shot but I wouldn't necessarily,you know, have a very high expectation of it working for any particular problem.



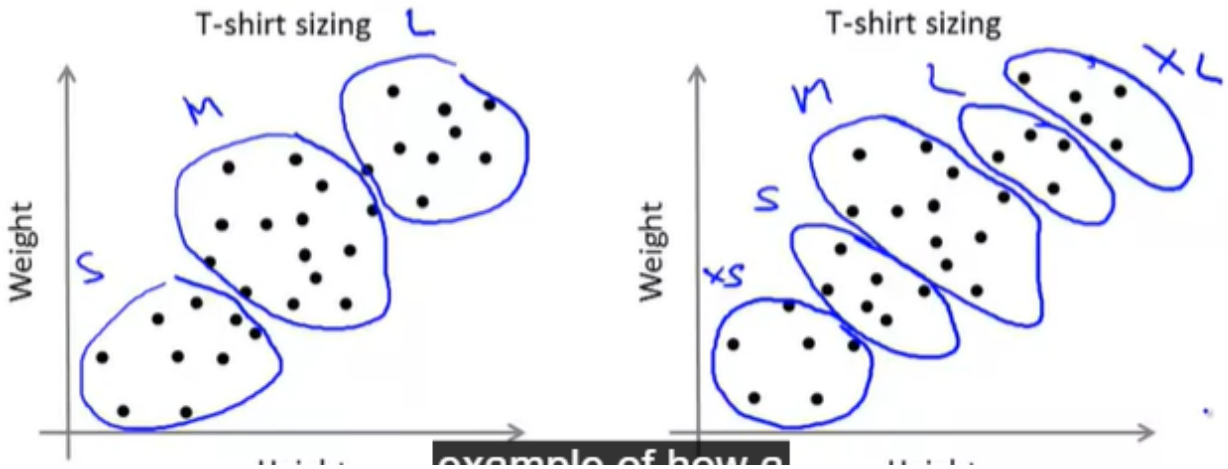
Choosing the value of K

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.

K=3 S, M, L

K=5 XS, S, M, L, XL

E.g.



Motivation 1 : Data compression

Dementionality reduction

second type of unsupervised learning problem called dimensionality reduction.

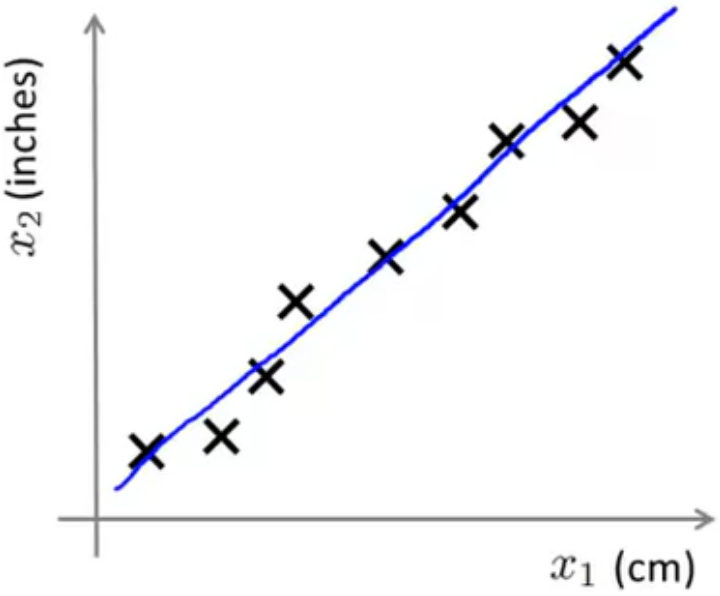
data compression not only allows us to compress the data and have it therefore use up less computer memory or disk space, but it will also allow us to speed up our learning algorithms

if the length in centimeters were rounded off to the nearest centimeter and lengthened inches was rounded off to the nearest inch. Then, that's why these examples

don't lie perfectly on a straight line, because of, you know, round-off error to the nearest centimeter or the nearest inch.

And if we can reduce the data to one dimension instead of two dimensions, that reduces the redundancy.

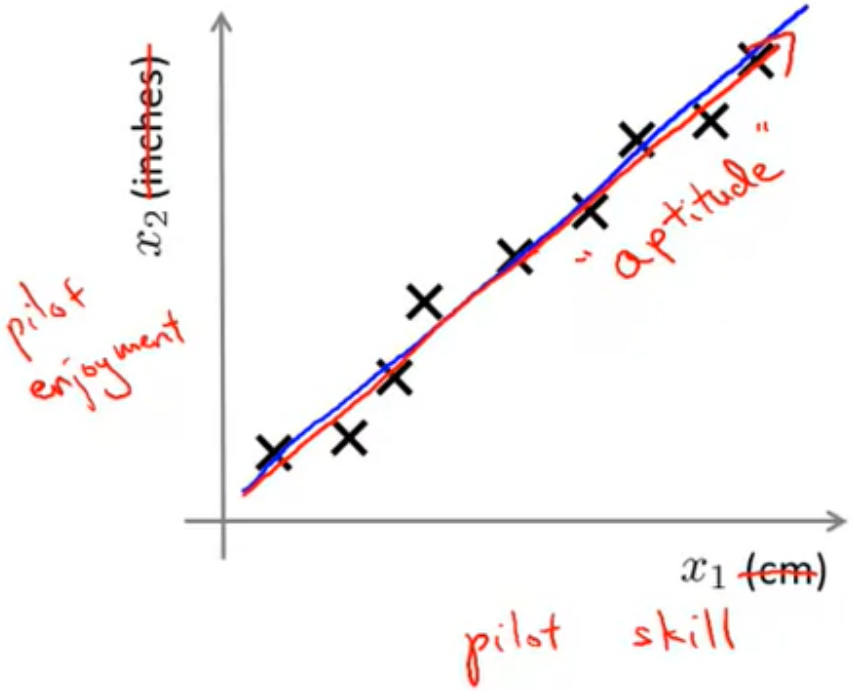
Data Compression



Reduce data from 2D to 1D

if you highly correlated features, maybe you really want to reduce the dimension

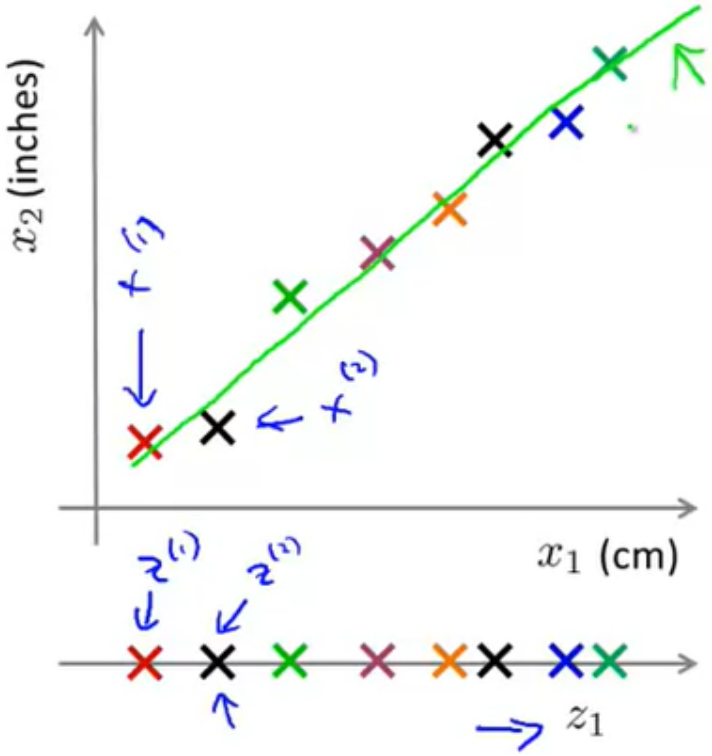
Data Compression



Reduce data from 2D to 1D

this is an approximation to the original training self because I have projected all of my training examples onto a line

Data Compression



Reduce data from 2D to 1D

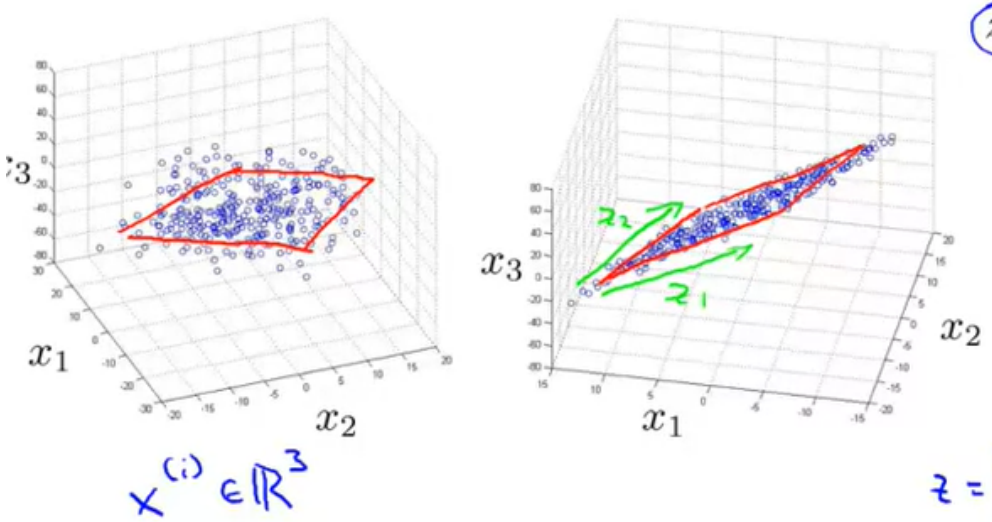
$$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$$

$$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$$

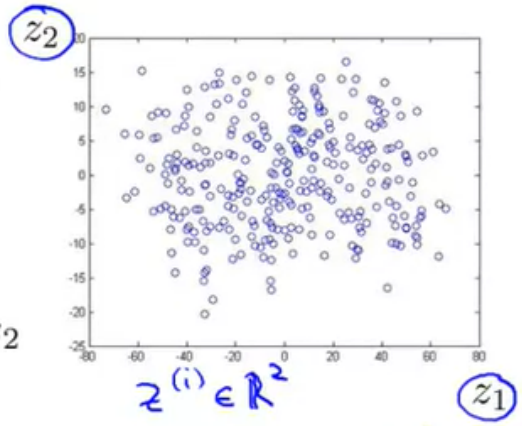
$$\vdots$$

$$x^{(m)} \rightarrow z^{(m)}$$

Reduce data from 3D to 2D



$x^{(i)} \in \mathbb{R}^3$



$z^{(i)} \in \mathbb{R}^2$

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

Motivation II : Data Visualization

Data Visualization

$x \in \mathbb{R}^{50}$

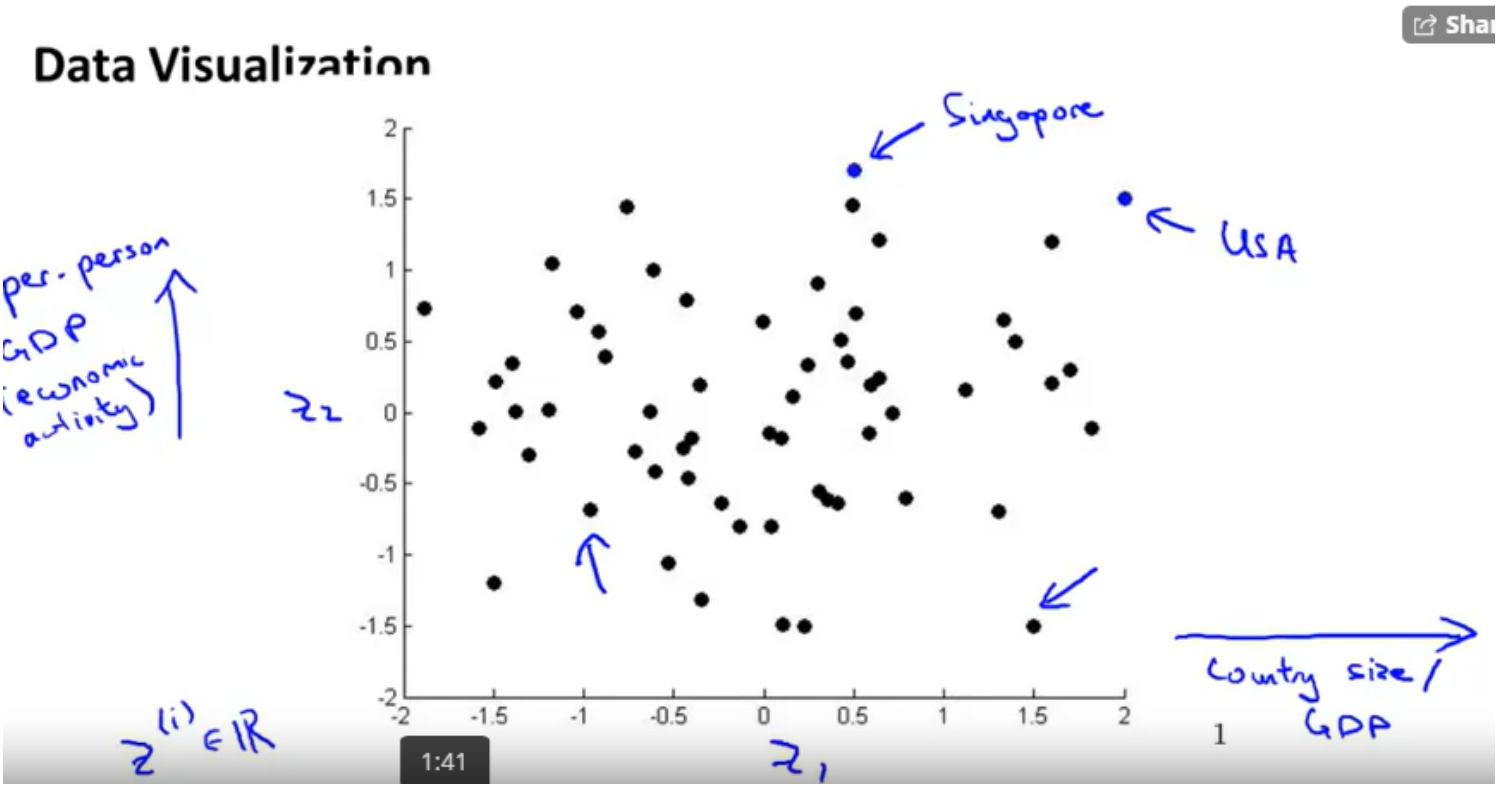
Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Develop-ment Index	x_4 Life expectancy	x_5 Poverty Index (Gini as percentage)	x_6 Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
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	...

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

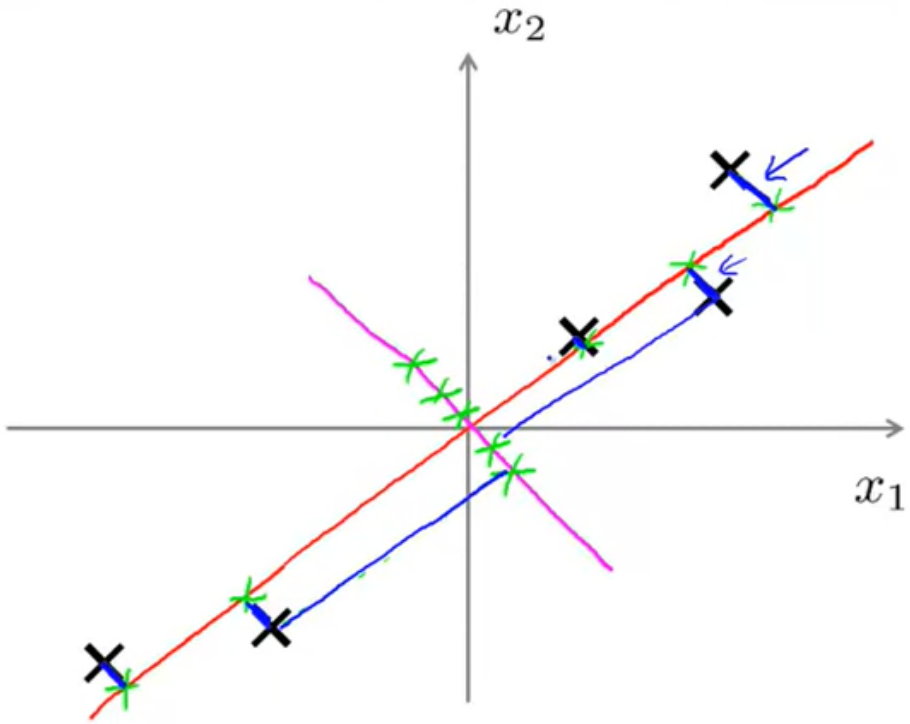


Principal Component Analysis Problem Formulation
PCA
For the problem of dimensionality reduction, by far the most popular, by far the most commonly used algorithm is something called
principle components analysis, or PCA

The length of those blue line segments, that's sometimes also called the projection error.
And so what PCA does is it tries to find a surface onto which to project the data so as to minimize that.

it's standard practice to first perform mean normalization at feature scaling so that the features x1 and
x2 should have zero mean, and should have comparable ranges of values.

Principal Component Analysis (PCA) problem formulation

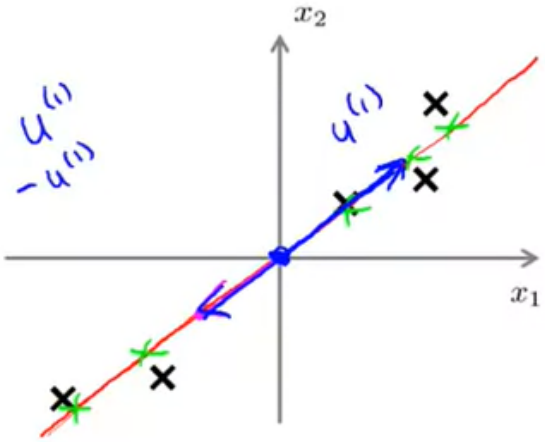


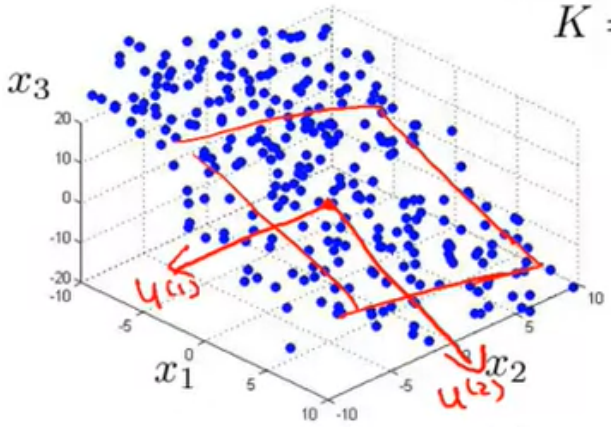
$x \in \mathbb{R}^2$

where the PCA gives me u(1) or -u(1), doesn't matter. positive vector or negative vector

Principal Component Analysis (PCA) problem formulation

$3D \to 2D$
$K = 2$





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.

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.

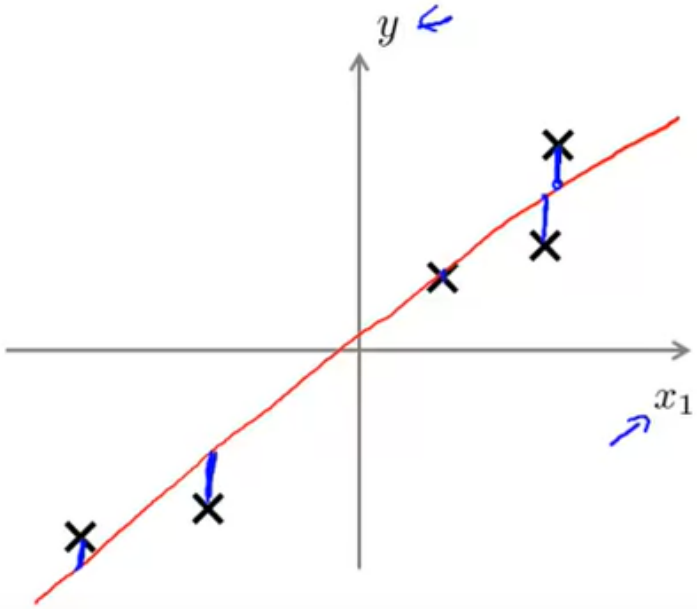
if you're not familiar with linear algebra, just think of it as
finding k directions instead of just one direction onto which to project the data.
So finding a k-dimensional surface is really finding a 2D plane in this case,
shown in this figure,
where we can define the position of the points in a plane using k directions.

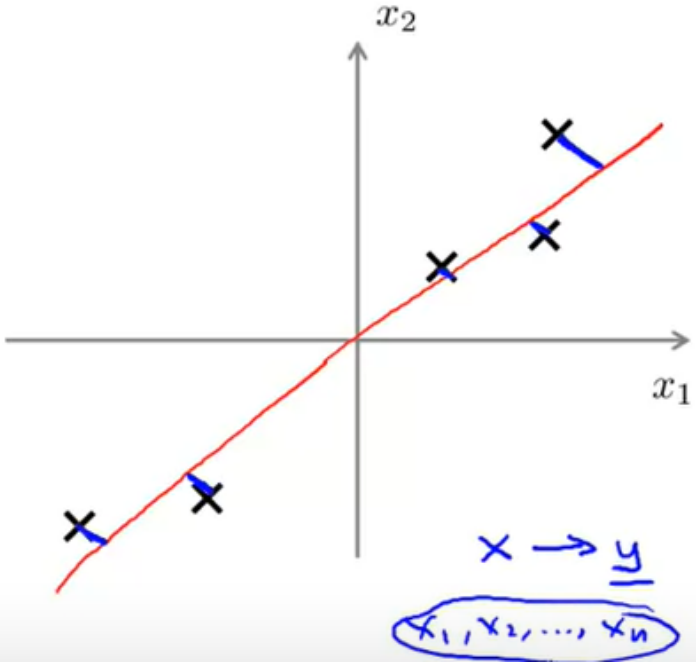
And that's why for PCA we want to find k vectors onto which to project the data.
And so more formally in PCA, what we want to do is find this way to project the data
so as to minimize the sort of projection distance,
which is the distance between the points and the projections.
And so in this 3D example too.
Given a point we would take the point and project it onto this 2D surface.
We are done with that.
And so the projection error would be, the distance between the point and
where it gets projected down to my 2D surface.
And so what PCA does is I try to find the line, or a plane, or whatever,
onto which to project the data, to try to minimize that square projection,
that 90 degree or that orthogonal projection error.

If we were doing linear regression, what we would do would be, on the left we would
be trying to predict the value of some variable y given some info features x.
And so linear regression, what we're doing is we're fitting a straight line so
as to minimize the square error between point and this straight line.
And so what we're minimizing would be the squared magnitude of these blue lines.
And notice that I'm drawing these blue lines vertically.
That these blue lines are the vertical distance between the point and
the value predicted by the hypothesis.
Whereas in contrast, in PCA, what it does is it tries to minimize
the magnitude of these blue lines, which are drawn at an angle.
These are really the shortest orthogonal distances.
The shortest distance between the point x and this red line.

there is no special variable y in PCA, every x1,x2, are treated the same

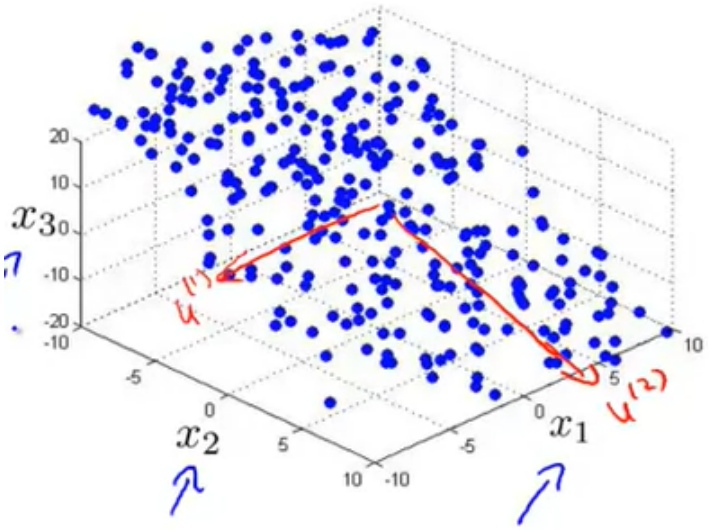
PCA is not linear regression

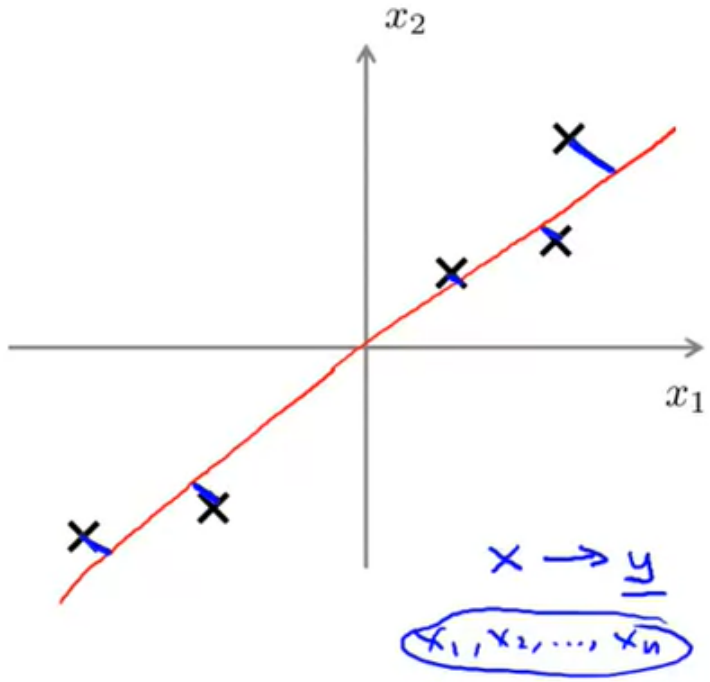




if I have three-dimensional data and
I want to reduce data from 3D to 2D, so maybe I wanna find two directions, u(1) and u(2), onto which to project my data.
Then what I have is I have three features, x1, x2, x3, and
all of these are treated alike.
All of these are treated symmetrically and
there's no special variable y that I'm trying to predict.

PCA is not linear regression





Principal Component Analysis algorithms

Before applying PCA, there is a data pre-processing step which you should always do.
Given the trading sets of the examples is important to always perform mean normalization,
and then depending on your data, maybe perform feature scaling as well.
this is very similar to the mean normalization and feature scaling
process that we have for supervised learning.

In fact it's exactly the same procedure except that we're doing it now to our unlabeleddata, X1 through Xm.
So for mean normalization we first compute the mean of each feature and then we replace each feature, X, with X minus its mean, and so this makes each feature now have exactly zero mean.

Similar to what we had with supervised learning, we would take x, i substitute j, that's the j feature

and so we would subtract of the mean, now that's what we have on top, and then divide by sj.
Here, sj is some measure of the beta values of feature j. So, it could be the max minus min value, or more commonly,
it is the standard deviation of feature j. Having done this sort of data pre-processing, here's what the PCA algorithm does.

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 $x_j - \mu_j$.

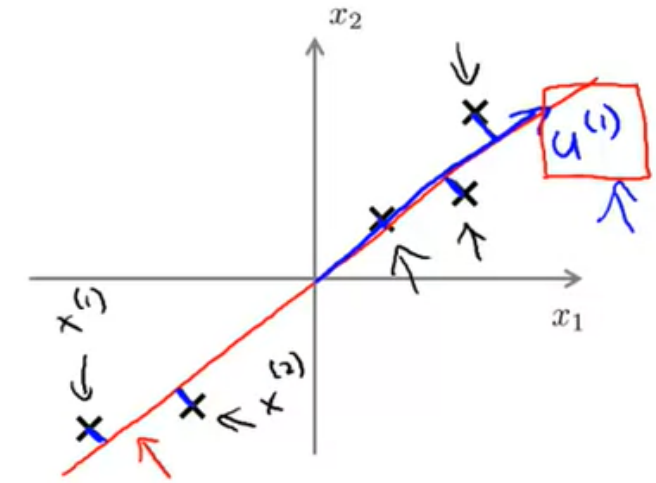
If different features on different scales (e.g., x_1 = size of house, x_2 = number of bedrooms), scale features to have comparable range of values.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

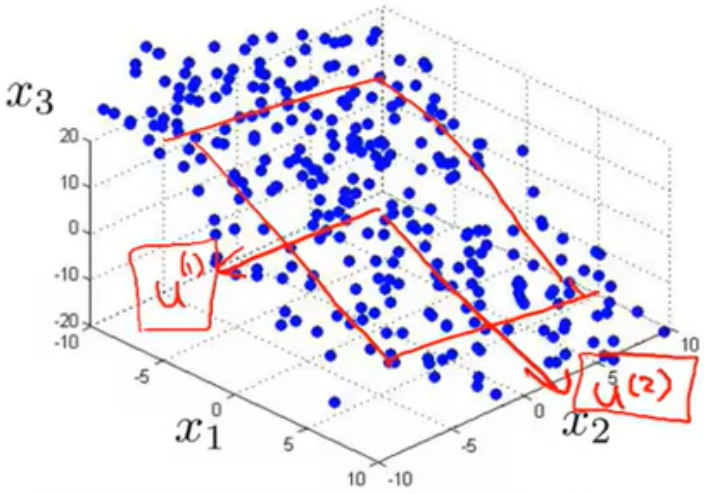
compute the vectors u(i) and compute the z values

https://www.youtube.com/watch?v=FgakZw6K1QQ StatQuest on PCA

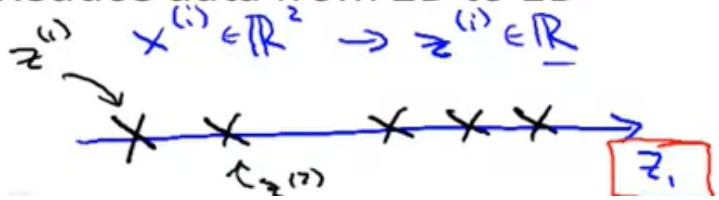
Principal Component Analysis (PCA) algorithm



Reduce data from 2D to 1D



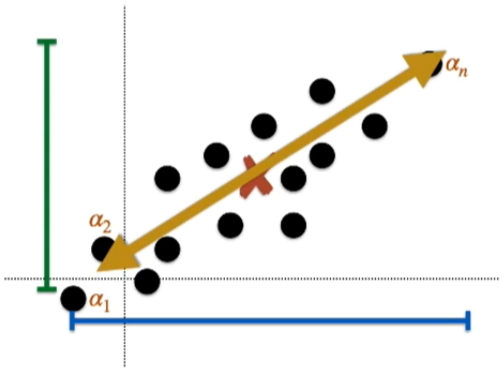
Reduce data from 3D to 2D



$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$

$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

Formulas



Weights	Points
α_1	(x_1, y_1)
α_2	(x_2, y_2)
$\vdots$	$\vdots$
α_n	(x_n, y_n)

Mean

$$(\mu_x, \mu_y) = \left(\frac{1}{\sum \alpha_i}\sum_{i=1}^{n} \alpha_i\, x_i, \frac{1}{\sum \alpha_i}\sum_{i=1}^{n} \alpha_i\, y_i\right)$$

x-Variance

$$var(x) = \frac{1}{\sum \alpha_i^2}\sum_{i=1}^{n} \alpha_i^2 (x_i - \mu_x)^2$$

y-Variance

$$var(y) = \frac{1}{\sum \alpha_i^2}\sum_{i=1}^{n} \alpha_i^2 (y_i - \mu_y)^2$$

Covariance

$$cov(x, y) = \frac{1}{\sum \alpha_i^2}\sum_{i=1}^{n} \alpha_i^2 (x_i - \mu_x)(y_i - \mu_y)$$

Covariance matrix

$$\Sigma = \begin{pmatrix} var(x) & cov(x, y) \\ cov(x, y) & var(y) \end{pmatrix}$$

It turns out that the SVD function and the I function it will give you the same vectors, although SVD is a little more numerically stable. So I tend to use SVD, although I have a few friends that use the I function to do this as wellbut when you apply this to a covariance matrix sigma it gives you the same thing.
This is because the covariance matrix always satisfies a mathematical Property called symmetric positive definite
You really don't need to know what that means, but the SVDand I-functions are different functions but when they are applied to a covariance matrix which can be proved to always satisfy this mathematical property; they'll always give you the same thing.

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$$

$n \times 1$ $\quad$ $1 \times n$ $\quad \rightarrow n \times n$

Sigma

Compute "eigenvectors" of matrix Σ:

→ Singular value decomposition

```
[U,S,V] = svd(Sigma);
```

eig (Sigma)

$n \times n$ matrix.

$$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(m)} \\ | & | & | & & | \end{bmatrix}$$

k

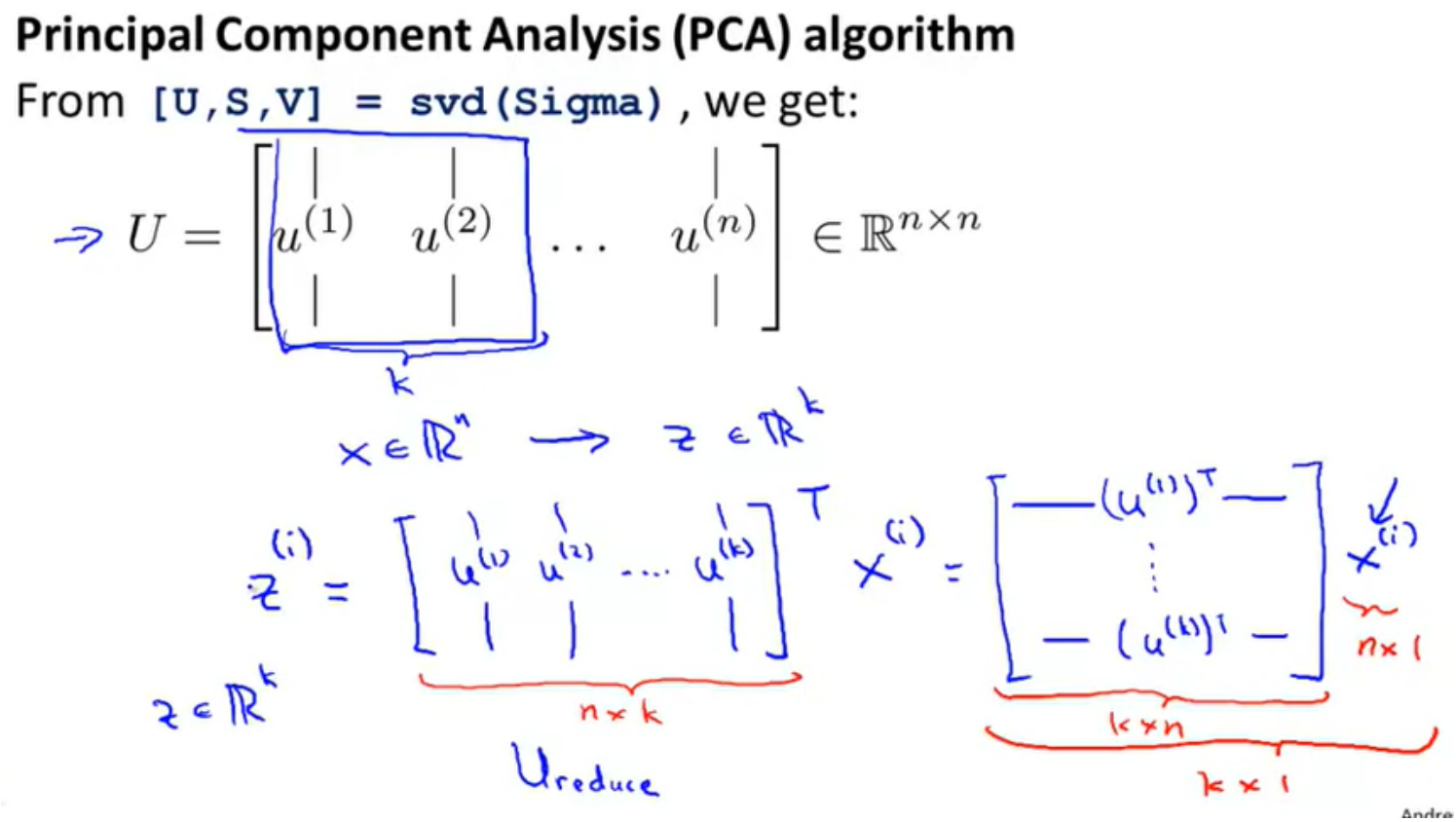
$U \in \mathbb{R}^{n \times n}$

$u^{(1)}, \ldots, u^{(k)}$

from the SVD numerical linear algebra routine we get these matrices u, s, and d. we're going to use the first K columns of this matrix to get u1-uK.
Now the other thing we need to is take my original data set, X which is an RN And find a lower dimensional representation Z, which is a R K for this data.
So the way we're going to do that is take the first K Columns of the U matrix.
Construct this matrix.
Stack up U1, U2 andso on up to U K in columns.
It's really basically taking, you know, this part of the matrix, the first K columns of this matrix.

And so this is going to be an N by K matrix.
I'm going to give this matrix a name.
I'm going to call this matrix U, subscript "reduce," sort of a reduced version of the U matrix maybe.
I'm going to use it to reduce the dimension of my data.

And the way I'm going to compute Z is going to let Z be equal to this U reduce matrix transpose times X. Or alternatively, you know,
to write down what this transpose means. When I take this transpose of this U matrix, what I'm going to end up with is these vectors now in rows. I have U1 transpose down to UK transpose.
Then take that times X, and that's how I get my vector Z



Principal Component Analysis (PCA) algorithm summary

After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\text{Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$$

$X = \begin{bmatrix} — (x^{(1)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$

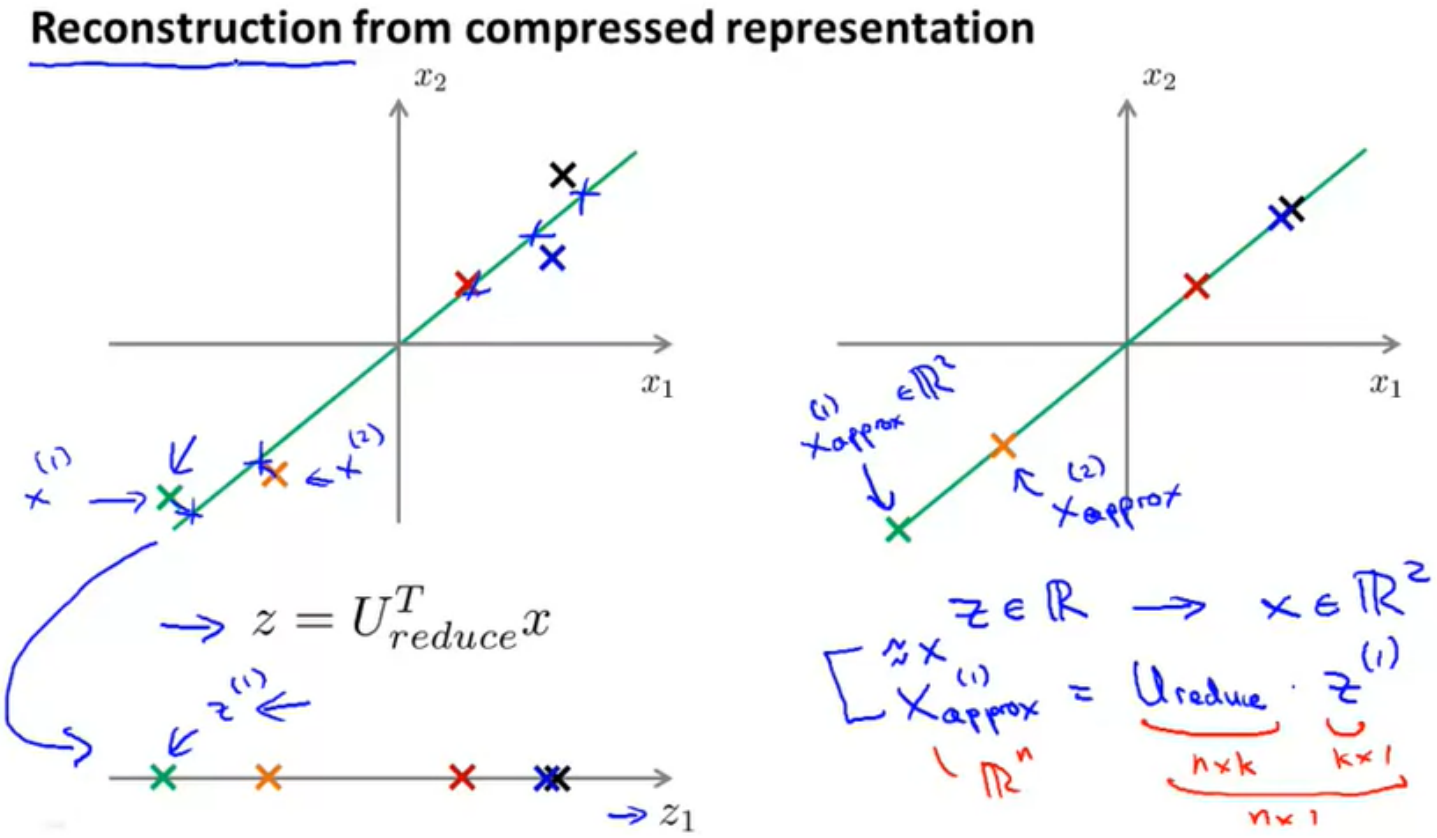
Sigma = (1/m) * X' * X;

```
[U,S,V] = svd(Sigma);
Ureduce = U(:,1:k);
z = Ureduce'*x;
```

$x \in \mathbb{R}^n$ $\quad$ ~~$x_0 = 1$~~

Applying PCA

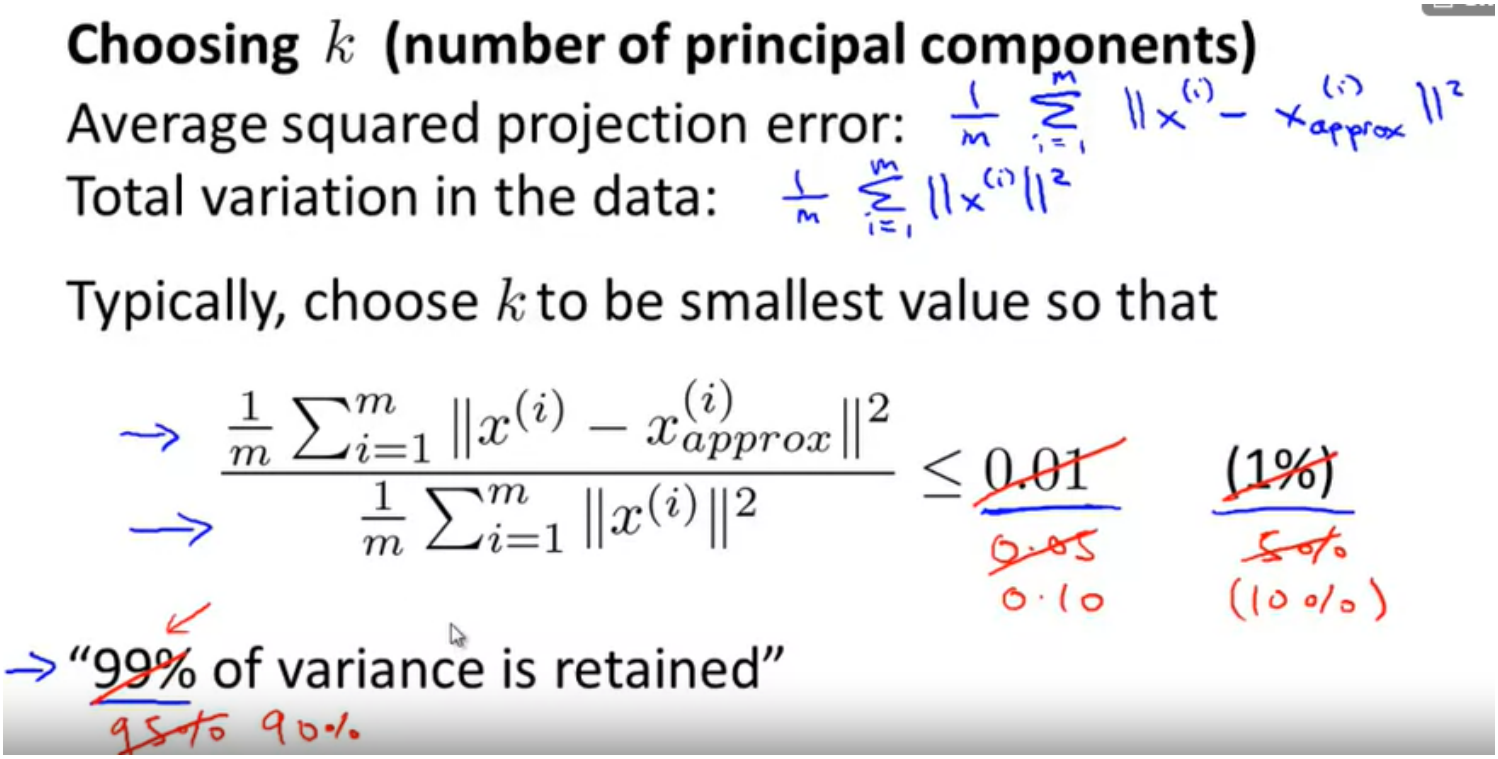
Reconstruction from compressed Representation

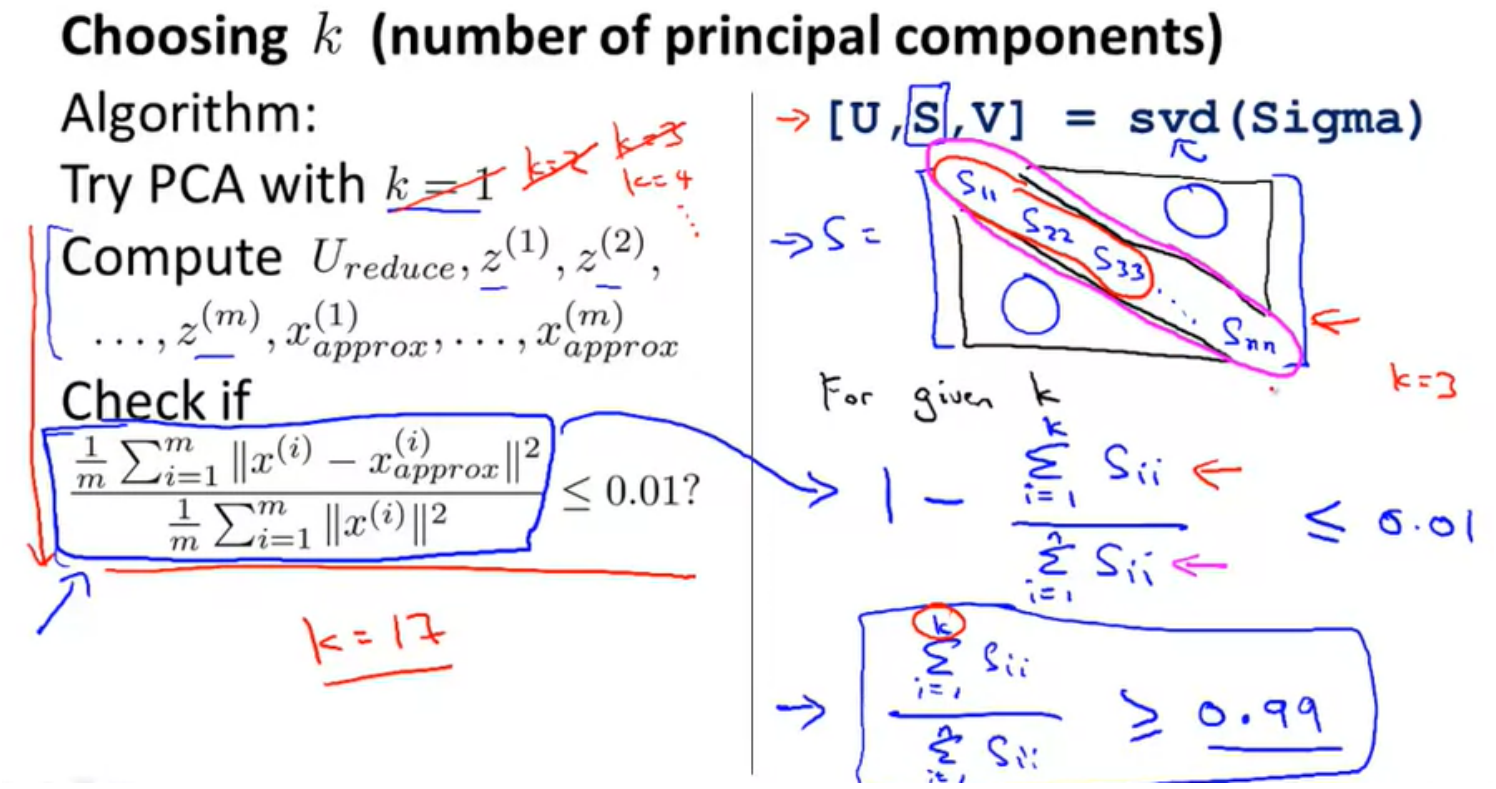


Choosing the number of principal components

In the PCA algorithm we take N dimensional features and reduce them to some K

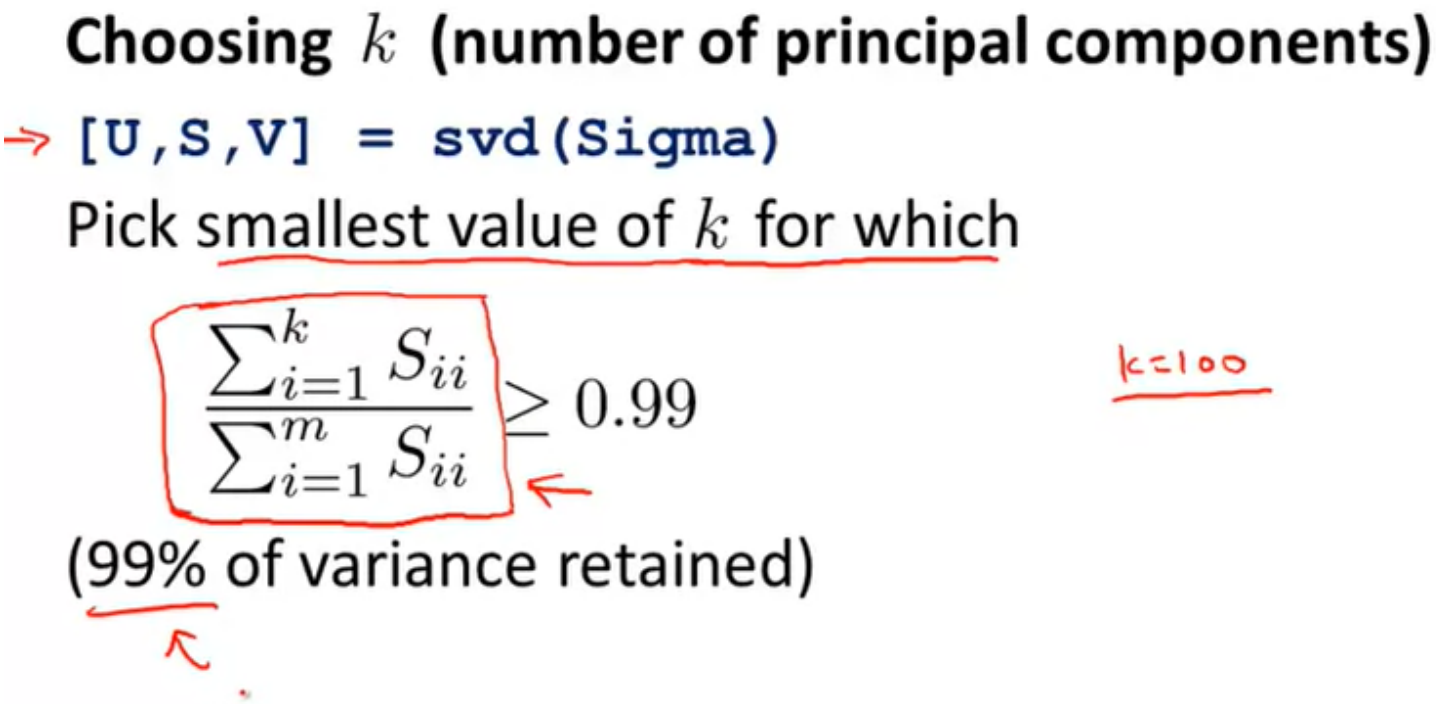
dimensional feature representation.
This number K is a parameter of the PCA algorithm.
This number K is also called the number of principle components or the number of principle components that we've retained.





And by the way, even if you were to pick some different value of K, even if you were to pick the value of K
manually, you know maybe you have a thousand dimensional data and I just want

to choose K equals one hundred.
Then, if you want to explain to others what you just did, a good way to explain the performance
of your implementation of PCA to them, is actually to take this quantity and compute what this is, and that will tell you what was the percentage of variance retained.
And if you report that number, then, you know, people that are familiar with PCA, and people can use this to get a good understanding of how well your hundred dimensional representation is approximating your original data set, because there's 99% of variance retained.
That's really a measure of your square of construction error, that ratio being 0.01, just gives people a good intuitive
sense of whether your implementation of PCA is finding a good approximation of your original data set.

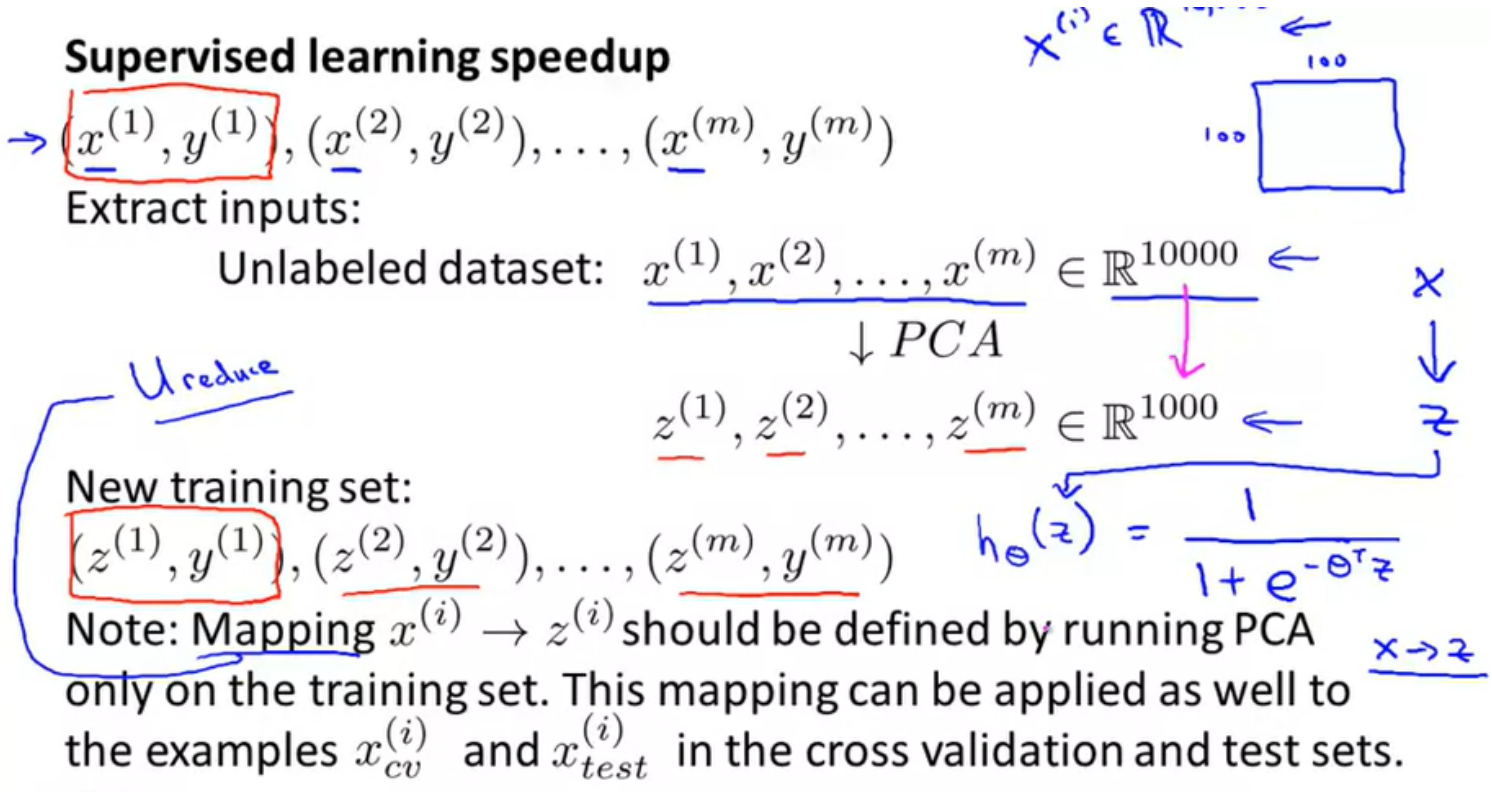


Advice for applying PCA

Here's how you can use PCA to speed up a learning algorithm, and this supervised learning algorithm speed up is actually the most common use that I personally make of PCA

One final note, what PCA does is it defines a mapping from x to z and this mapping from x to z should be defined by running
PCA only on the training sets. And in particular, this mapping that

PCA is learning, right, this mapping, what that does is it computes the set of parameters.
That's the feature scaling and mean normalization. And there's also computing this matrix U reduced. But all of these things that U reduce, that's like a parameter that is learned by PCA and we should be fitting our parameters only to our training sets and not to our cross validation or test sets and so these things the U reduced so on, that should be obtained by running PCA only on your training set.
And then having found U reduced, or having found the parameters for feature scaling where the mean normalization
and scaling the scale that you divide the features by to get them on to comparable scales.
Having found all those parameters on the training set, you can then apply the same mapping to other examples that may be
In your cross-validation sets or in your test sets.



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 $k = 3$

Bad use of PCA: To prevent overfitting

Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$. — 1000 — 10000

Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

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

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 $x^{(i)}_{test}$ to $z^{(i)}_{test}$. Run $h_\theta(z)$ on $\{(z^{(1)}_{test}, y^{(1)}_{test}), \ldots, (z^{(m)}_{test}, y^{(m)}_{test})\}$

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 $z^{(i)}$.

Week9

Density Estimation

Anomaly detection : Problem Motivation

Anomaly detection example

Aircraft engine features:

- x_1 = heat generated
- x_2 = vibration intensity
- ...

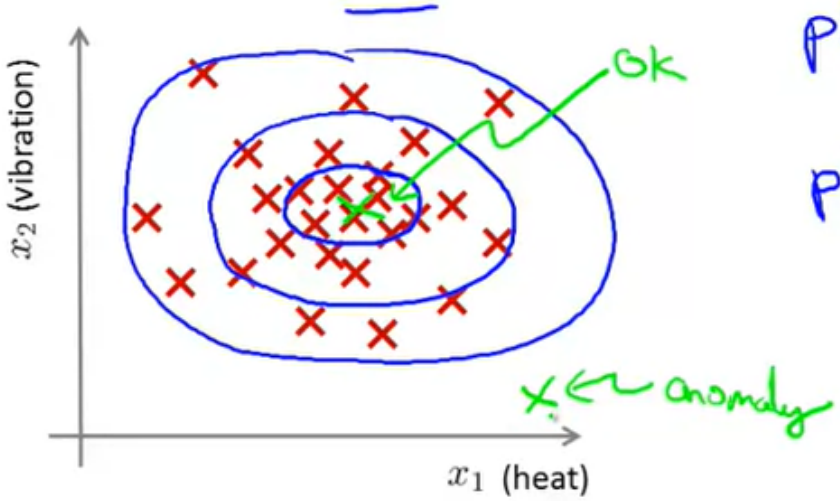
Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

New engine: x_{test}

Density estimation

- Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
- Is x_{test} anomalous?

Model $p(x)$.



$p(x_{test}) < \varepsilon \rightarrow$ flag anomaly

$p(x_{test}) \geq \varepsilon \rightarrow$ ok

Week10

Week11

Octave

Matrices And vectors

```
% 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)
```

Addition and Scalar Multiplication

```
% 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
```

Matrix-Vector Multiplication

```
% Initialize matrix A
A = [1, 2, 3; 4, 5, 6;7, 8, 9]
% Initialize vector v
```

```
v = [1; 1; 1]
% Multiply A * v
Av = A * v
```

Matrix-Matrix Multiplication

```
% Initialize a 3 by 2 matrix
A = [1, 2; 3, 4;5, 6]
% Initialize a 2 by 1 matrix
B = [1; 2]
% We expect a resulting matrix of (3 by 2)*(2 by 1) = (3 by 1)
mult_AB = A*B
```

Multiplication Properties

```
% 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
```

Inverse and Transpose

```
% Initialize matrix A
A = [1,2,0;0,5,6;7,0,9]
% Transpose A
A_trans = A'
% Take the inverse of A
A_inv = inv(A)
% What is A^(-1)*A?
```

```
A_invA = inv(A)*A
```

When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv.' The 'pinv' function will give you a value of \theta$\theta$ even if X^TXX^TX is not invertible

```
a
a;
a=pi
disp(a);
disp(sprintf('2 decimal : %0.2f' ,  a ))
format long
format short
a = [ 1 2; 3 4; 5 6]
a = 1 2
          3 4
          5 6
v = [1 2 3]
v = 1 2 3
v = 1:0.1:2
1.0000 1.1 1.2 1.3 .... 2.0
v = 1:6
1 2 3 4 5 6
ones(2,3)
zeros(1,3)
rand(1,3)
randn(1,3)
hist(w) - plot
hist(w,50)
eye(5)
size(A)
size(A,1) - row
size(A,2) - column
length(v) - length of longer dimension
```

Linear Algebra

Matrices and Vectors

Matrices are 2-dimensional arrays
A vector is a matrix with one column and many rows
So vectors are a subset of matrices.
Notation and terms:
• A_{ij}*Aij* 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*vi* 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}R refers to the set of scalar real numbers.
• \mathbb{R^n}R*n* refers to the set of n-dimensional vectors of real numbers.

• Matrices are not commutative: A∗B \neq B∗A*A*∗*B*=*B*∗*A*
• Matrices are associative: (A∗B)∗C = A∗(B∗C)(*A*∗*B*)∗*C*=*A*∗(*B*∗*C*)

The **inverse** of a matrix A is denoted A^{-1}*A*−1. Multiplying by the inverse results in the identity matrix.
A non square matrix does not have an inverse matrix. We can compute inverses of matrices in octave with the pinv(A)*pinv*(*A*) function and in Matlab with the inv(A)*inv*(*A*) function. Matrices that don't have an inverse are *singular* or *degenerate*.
The **transposition** of a matrix is like rotating the matrix 90° in clockwise direction and then reversing it.

Examples: $m = 4$.

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	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

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

m x (n+1)

m-dimensional vector

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