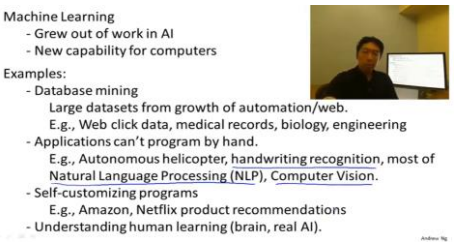
W1-Instrcution

1. Welcome

1.1 Machine learning_example:
. You probably use it dozens of times a day without even knowing it:
>1. Each time you do a web search on Google or Bing, that works so well because their machine learning software has figured out **how to rank what pages;**
>2. When Facebook or Apple's photo application recognizes your friends in your pictures, that's also machine learning;
>3. Each time you read your email and a spam filter saves you from having to wade through tons of spam, again, that's because your computer has learned to distinguish spam from non-spam email.

1.2: Machine learning defination:
A science of getting computers to learn without being explicitly programmed.
e.g: getting robots to tidy up the house: having the robot watch you demonstrate the task and learn from that. The robot can then watch what objects you pick up and where to put them and try to do the same thing even when you aren't there.

1.3: AI application: reasons excited about AI, or artificial intelligence problem, is **building truly intelligent machines**, we can do just about anything that you or I can do.
Many scientists think the best way to make progress on this is through learning algorithms called **neural networks, which mimic how the human brain works**, and I'll teach you about that, too. In this class, you learn about machine learning and get to implement them yourself.



2. Instruction on what is MT

2.1 MT: older definiaton: field of study give computes the ability to learn witout being explicityly programmed.

Checker player algorithm by Arthur Samuel: algorithm play tens of thousands of games against itself, and by watching what sorts of board positions tended to lead to win/looses, the checker palyser algorithm learned over time what are good boad postion and bad positions.-->learned to play checker better.
note: computer have patient to play ten of thousand to learn experiance, while human have no such patience.

2.2 MT latest defination: a computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T , as mearused by P, improves with experience E.

2.2: MT algorithms

- **Supervised learning:** teach computer how to do something;

- **Unsupervised learning:** computer learn by itself.
- Others: reinforcement learning, recommender systems.

Need to pay attention: Practical advice for applying learning algorithms.

Machine Learning definition

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Machine learning algorithms:
- Supervised learning
- Unsupervised learning

Others: Reinforcement learning, recommender systems.

Also talk about: Practical advice for applying learning algorithms.

3. Supervised learning_Insturciton

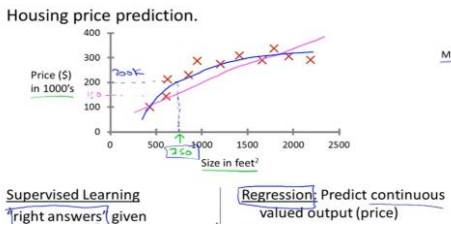
3.1 Supervised learning defination:
Give the algorithm a data set, in which 'right answers' were given, **task of algorithm was to produce more of these right answers.**
in every sample in our data set, we are told what is the 'correct answer', that we would have quite liked the algorithm have predicted on that example.

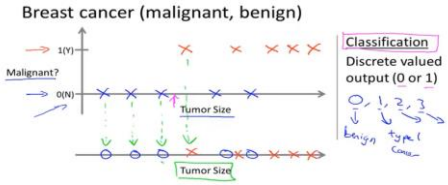
3.2 Regression problem: trying to predict continuous valued output （连续值输出）
3.3: Classification problem: trying to predict a discrete valued ouput （离散值输出, e.g: 0 or 1）
Note: classificaiton could have more than two possible values for the output

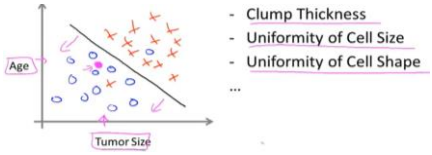
e.g:
- house price prediction; supervised learing, also regression problem
- Breast cancer: supervised learing, classficaiton problem:
  - in this case: use only feature (tumor size) to define classfication type;
  - have different plot draw to show: two axile (x: tumor size, Y, output 1, 0) with same shape plots; and only 1 axile (turmor size) with different plots represnt different result.
- brease cancer : two feature (tumor size and age); could have more feature;

Note: learning algorithm can deal with not only 1, 2, 3 features, but an **infinite number of features.**

with infinite features, computer will run out of memeory--->support vector machine: there will be neat matchmatical trick, allow computer to deal with an infinite number of features. How -->kernal, then have m features.



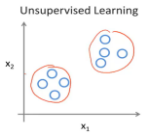




4. unsupervised learning_Insturciton

4.1 training data:
supervised learning : training data have right answere;
unsupervised learning: training data doesn't have any labels , **or all has the same label** (e.g: anomaly detection), or really no labels.

4.2: fucntion , task:
we are given a datat set, and not told what to do with it, not told what each data point is.--> just told , here is a data set, can you find some structure in the data?

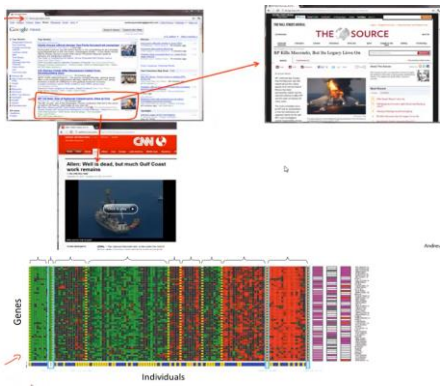


4.3 Clustering Algorithm

- Clustering algorithm
Algorithm break data set into two/more separate clusters.

Application e.g:
1. google news (news.google.com): everyday google news goes and look at hundreds of thousands of new stories on the web, and it groups them into cohesive news stories: automatically cluster them together, so the new stories that all about the same topic get displayed together;

2. Understanding genomics:
- Given DNA microarry data: have a group of different individuals, and for each of them, measure how much they do or do not have a certain gene-technically mesure how much certain genes are expressed.
colors red, green, gray and so on, show the degree to which different individuals do or do not have a specific gene.
- Run clustering algorithm to group individuals, into different categories or into different types of people.
This one is unsupervised learning program as do not tell in advance the 'right type' to certain genes. Instead only saying here is a bunch of data, do not know what's the data, do not know who is in what type, even do not know what the different types of people are, -->can you/algorithm automatically find structure in the data, automatically cluster the individuals into these types that i do not know in advance?



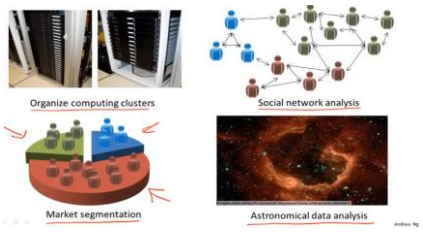
4.4 Applications_clustering algorithm

- Organize large computer clusters: try to find out which machines tend to work together (if put them work together, can make data center work more efficiently)

- Social network analysis: Given knowledge about which friends you email/facebook / circle friends the most, automatically classify which are cohesive groups of friends, which are groups of people that all know each other.

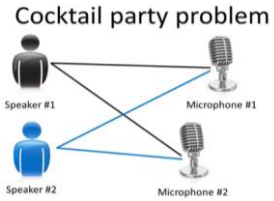
-Market segmentation: have huge databases of cutomer informaton, and automatically discover market segments and automatically group customers into different market segments. (do not know in advance market segments and customer group)

-Astronomical data analysis: clustering algorithms give usefuel theory of how galaxies are formed.



4.5 Cocktail party problem

1. e.g:
-two people in a room, talking at the same time;
- two microphones in the room: each microphe at different distance from two people.
- Each microphone records: a different/overlapping combination of these two speaker voices (mayber speaker 1 is a little bit louder on microphone1, and speaker 2 louder in microphone 2).
-give two microphone recordings to an unsupervised learning algorithm-cocktail party algorithm, and tell algorithm find structure in this data.
--cocktail part algorithm: listen to these audio recordings and say it sounds like two audio recordings that are being summed together; and alogrithm will separate out these two audio sources that being summed together to form other recording.



2. Code_cocktail party problem
only one line code: [W, s, v]=svd((repmat(sum(x.*x1, size(x,1),1).*x)*x');

svd function: singular value decomposition (奇异值分解）.turn out to be a linear algebra routine (线性代数常规函数） that is just built into Octave.

If do with c++o or Java or python, need many codes linking complex C++, or Java libraries. much more complicated to do so in those languages.

Cocktail party problem algorithm

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

4.6 Octave programming enviroment
use in MT programming;
Octave:free open source software.
Use Octave or Matlabe many learning algorithm become just a few lines of code to implement.

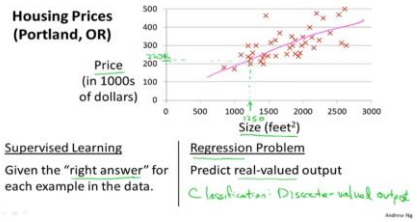
Normal process:
1. First prototype learning algorithm/ software in Octave;
Because software in Octave makes it incredibly fast to implement these learning algorithms.
2. Only after gotten it to work, then might migrate it to C++ or Java or whatever.

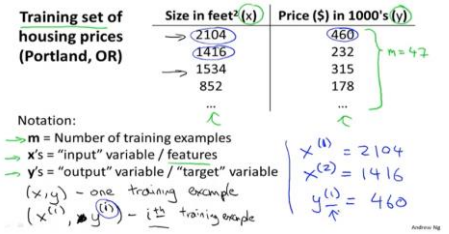
--->In this way much faster than if starting out in C++.

W2-Linear regression with one variable

1. Model-representation_Linear rgerssion with one variable

1.1 Note:
m=number of training examples
x's='input' variable / features
y's='output' variable / 'target' variable
(x, y)=one training example
(x(i), y(i))= i th training example

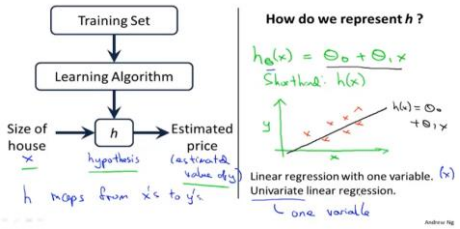




1.2 Estimation process

train model: training set--> learning algorithm---->prediction fucntion h
predict with trained h: new input--->h----->prediciton y

hypothesis h: maps from x's to y's



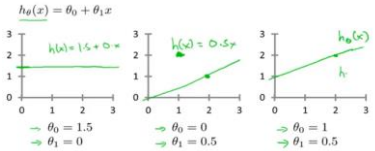
1.3 hypothesis h:-Linear regression
predict h is linear function: h_θ(x)=θ_0 + θ_1 *x

could be more complicated non-linear function, just start with this example first of fitting linear function, and build on this eventually more complex models and more complex algorithms.

2. Cost funtion_Linear regression with one variable

2.1: Paramter: θ_0, θ_1
choose paramter to make prediction function h correspond to a **good fit to the training set** (prediction close to training set).

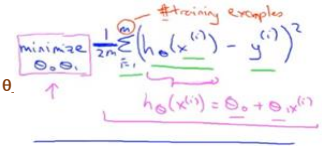
IDEA: Choose parmaters at least fit training set: give the x's in the training set, make **reasonable accurate predictions** for the y values.

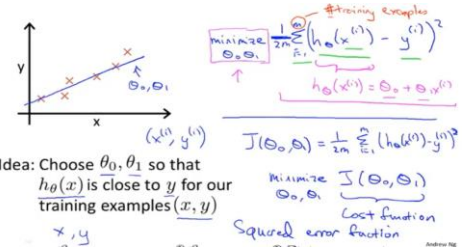


2.2 COST fucntion

mininize square of(prediction output- label output), over θ_0, θ_1 .
1/m-->average square, 1/2m: do not change object, just makes some of the math a littler easier..

Overall objective function-Squared error function:
finding paramter θ_0, θ_1 minimize the cost J(θ_0, θ





Note: there are other mehtod cost function, but square error function is the most commonly used.

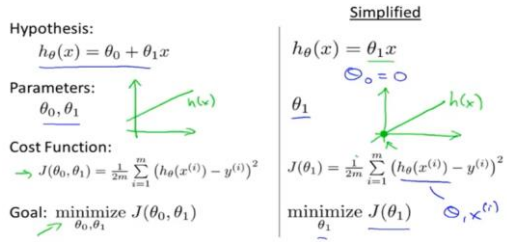
3. Cost function intuition_linear regression with one variable

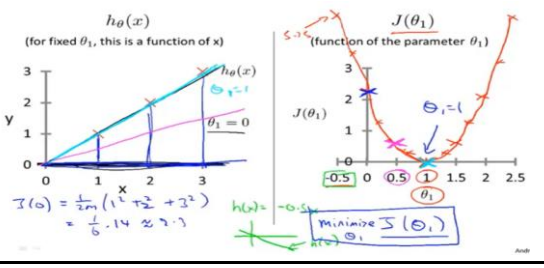
Prediciton function h_θ (x): for fixted θ, this is function of x

Cost function J(θ): function of the paramter θ.
In ploting pic, horizental axile is labled paramter θ.

Summary:
- each paramter θ_(i) corresponds to a diffrent prediciton function: line fit;
- each prediciton fit line, get the cost result J(θ_(i))
- plot pic based on pair (cost result J, θ)

- Object: choose paramter, minimize cost J (corresponding to find a sraight line that fit training set well) .

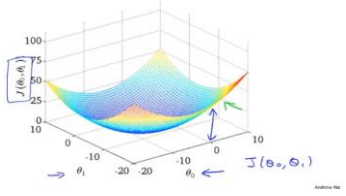




4. Cost function intuition II_linear regression with one variable

4.1 3D Visualize cost function: with all paramters (θ_0,θ_1)

Note: cost function is bowl shape over each parameter θ_0, θ_1 seperately. and **still bowl shape over paramter pair** (θ_0, θ_1) at same time.
E.g : right 3D, high is J.

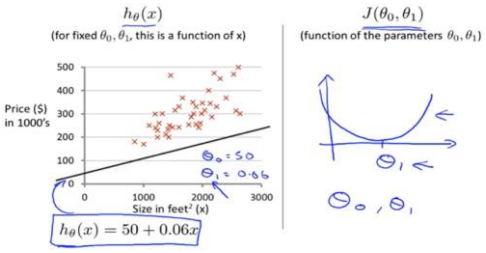


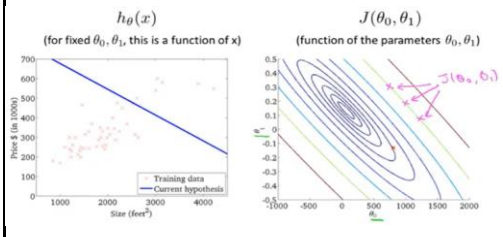
4.2 Contour plots Visualize cost function: with all paramters (θ_0,θ_1)

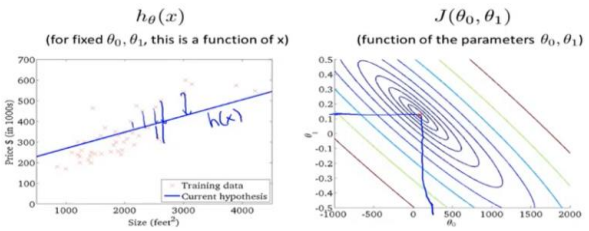
Contour plots: axis are paramter θ_0 and θ_1, each of these oval/ellipses show a set of points that takes on the same value for cost J (θ_0,θ_1)

Like: Project the 3D bowl to 2D paramter space vertically, the middle small circle of bowl bottom, each oval corresponds to same high postion in bowl (same cost J).

understanding: cost function value J how corresponds to different hypothesis h, and how better hypotheses may corresponds to points that are closer to the minimum of this cost function J.







5. Gradient descent_ Linear regression with one variable

Gradient descent used in all ML, and for cost function with as many number paramters.

5.1 Out line_gradient descent:

- start with initial paramter θ: whatever value can be is ok
- keep changing paramters a little bit, to reduce cost function value J(θ), until hopefully end up at a minimum, or maybe a local minimum.

E.g:
standing at one point on the hill, and look all around (360 degree), find the best direction to take a little step downhill,
-->at new point on the hill, find a direction in order to take a little baby step downhill.
--> and so on, until converge to local minimum

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

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

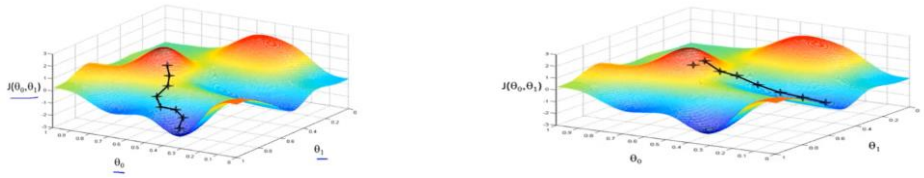
Outline:

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

until we hopefully end up at a minimum

Note: if take different initalizting point:
repeate above step: take a little step in the direction of steepest descent, --->so on, **may lead to the second local optimum, right pic**

slightly different initial paramter may lead to very different local optimum.



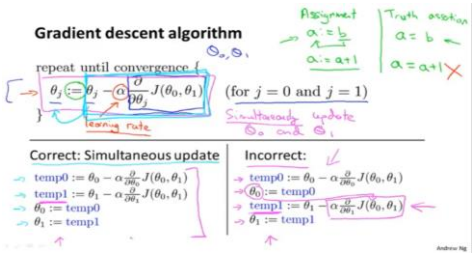
5.2: Gradient descent algorithm

'a:=b' : assignment: take b and use it to overwrite whatever the value of a;
'a=b': true assertion /claming-->1 true, 0 fail
a: learning rate: how big step take downhill with gradient descent. a large-->very aggressive gradient descent procedure, a small-->little baby steps downhill

repeat until convergence {

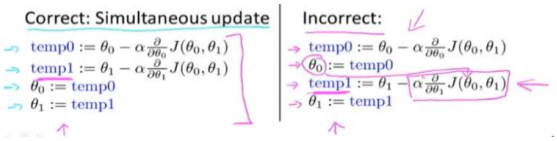
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ (for $j = 0$ and $j = 1$)

}



Note: simultaneously at the same time update all the parameters in gradient descent
more natural to implement the simultaneous update, incorrect update (unsimultaneous) in below might work , but have strange properties, not commonly use

e.g: in the right side, when calcualting θ_1, new upodated θ_0 has been used in derivative of J(θ_0, θ_1) over θ_1.



6. Gradient descent intuition_linear regression with one variable

1. Derivative J'(θ_i)
θ=θ-a***J'(θ_i)**-->
1.1. J'(θ_i) >=0----->θ will decrease, direction to reduce J;
1.2 J'(θ_i) <0----->θ will incrase, also direction to reduce J;

-J'(θ_i), make sure θ change in gradient descent , **always in direction of reducing J.**

2. Learnign rate

- a too small, gradient descent can be slow.
  To take a lot of mini baby steps to get to the minimum.

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

Gradient descent algorithm

repeat until convergence {

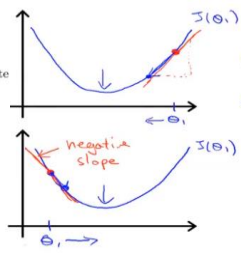
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

}

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

learning rate — derivative

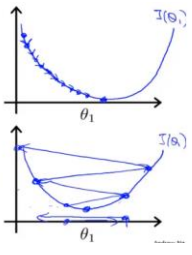
$\min_{\theta_1} J(\theta_1)$ $\theta_1 \in \mathbb{R}$.



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

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

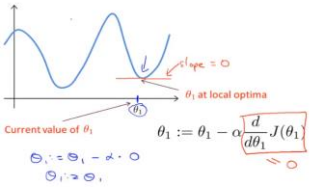
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



3. Gradient descent in local minimum

if parameter is already at a local minimu, one step of gradient descent actually does nothing, does not chagne parameter, J'(θ_i)=0.

this is good as keep solution at local optimum-local optimum is good enough?



4. Gradient descent can converge to a local minimum, even with the learing rate a fixed

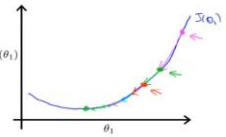
as approach a local minimum, gradient descent will automatically take smaller steps (derivative smaller approaching to 0). So, no need to decrease learning rate over time.

derivative will become smaller and smaller (less steep) as approching to the local optimum which derivative is 0.-->graident step: a*derivative become smaller.
when parameter take one step of gradient step, in the new postion, the new gradient is less steeper.

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

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

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



Gradient descent can use to decrease any function J, not only linear regression.

7. Gradient descent for linear regression

1. Linear regression cost function

Gradient descent algorithm

repeat until convergence {

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

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

}

Linear Regression Model

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

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

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

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

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

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

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

2. Gradient descent algorithm

update all paramters θ_i simultaneously

Gradient descent algorithm

repeat until convergence {

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

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

}

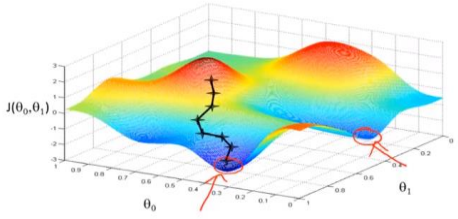
update θ_0 and θ_1 simultaneously

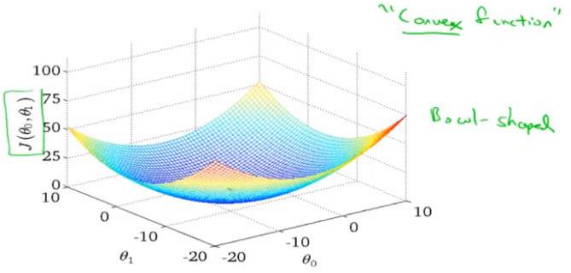
Note: loss function defination:factor 1/2m--->for getting right derivative

derivative defined: 1/m * sum of each example loss l(i) 's derivative over parameter: = 1/m * [d_l(i) /d_θ : sum over i] (so as to make overall cost of all examples decrease)
-->total cost function using factor 1/2m --> to get right derivative representation:

3. Convex function_linear regression function

susceptible to local optimum （容易陷入局部最优）: different intializaiton may endup different local optimum
yet cost function of linear regression is always bowl-shaped-**convex function**: does not have any local optima, except one global optima.
Gradient descent for linear regression cost function, always converge to global optima

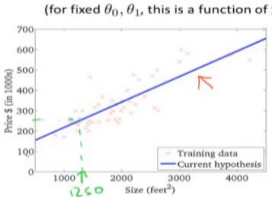




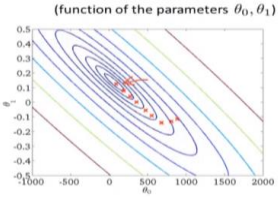
4. Linear regession cost function-gradient descent

global optima corresondings to best fitt hypothese h to training set.

$h_\theta(x)$ (for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$ (function of the parameters θ_0, θ_1)



5. 'Batch' gradient desceth

above gradient descnet also call batch descent, refere to the fact, **in every step of gradient descent, we are looking at all of the training examples:**
calcuate derivative: J'(θ). **J(θ)=1/(2m) * sum of all examples loss J_i**. go through all training examples in every step derivative calculation (y^_i - y_i = h(x_i) - y_i)-->j_train decrease in every step (note: if learning rate is reasonable-no overshooting)

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

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

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

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

"Batch" Gradient Descent

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

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

6. Mininum cost function:
normal equiation method: numerically solve the minimum issue of cost funciton.
Yet gradient descent scale better to large data set, than normal equations method.

Note: derivative dJ/dθ (θ derivative vector) = X . (ΘT. X - Y) T
X= (x1, x2, ..xm)
Y = (y1, ···ym)
make dJ/dθ =0-->get θ

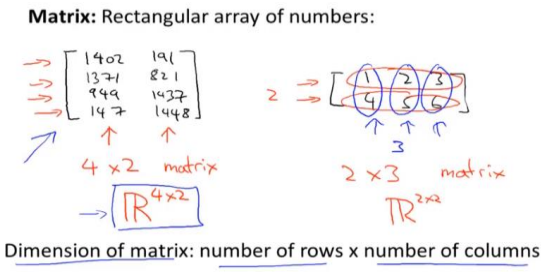
W3-Linear Algebra review (线性代数回顾)

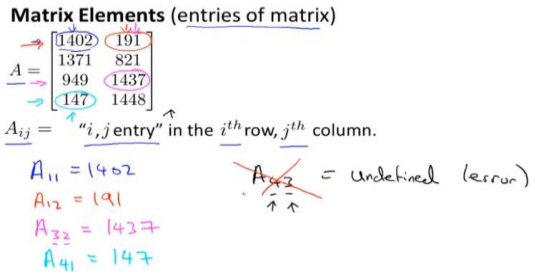
3_14. Matrics and vectors _Linear Algebra review

1. Matrix: Rectangular array of numbers; use uper case for matrix (e.g.: A, B, ..)

Dimension of matrix: number of rows x number of columns

2. Matrix Elments:
Aij = 'i,j entry' in i_th row, j_th colum



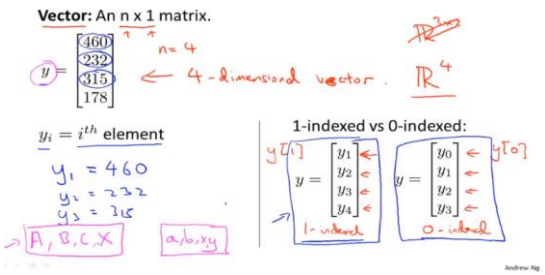


3. Vector: n x 1 matrix (special matrix: only one columen) ; **use low case for vector (e.g. a, b c)**
vector space: R_n (n: vector dimension number)

4. Vector element: yi = i_th element

5. Vector index: could start from 1 or 0
start from 1: y=[y1, y2, ..yn]; (more common in match)
start from 0: y=[y0, y1, ..yn-1] (more common in learning algorithm)

unless otherwise specified:
use 1-index vector throughout rest of videos on linear algebra review.
use 0-index in machine learning applicaitons

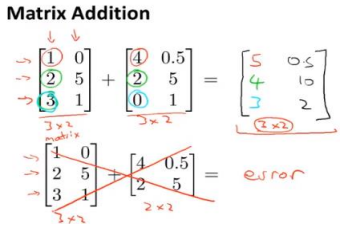


3_15. Additon and scalar multiplication _Linear Algebra review

2.1 Matrix Addition

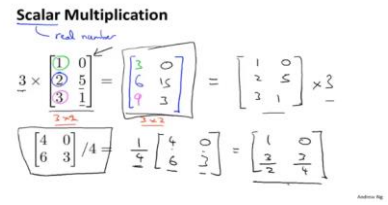
method: add up the elements of these matrices one at a time
C=A+B-->
C_ij = A_ij + B_ij

note: only same dimension matrices could add together.

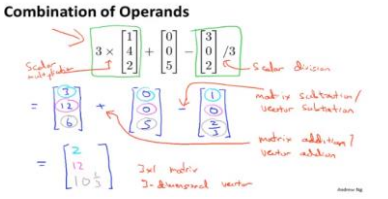


2.1: Scalar multiplication
scalar: a real number

method: each element of matrix muliply with scalar seperately.
result dimenson: have same dimension with matrix muliplied



2.3: Combination of operands.



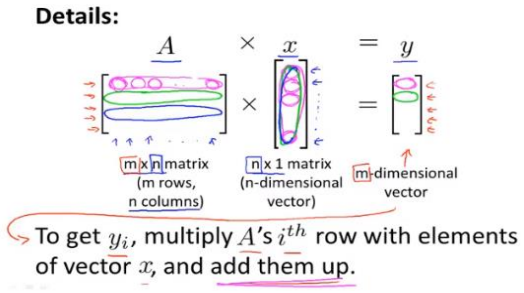
3_16. Matrix-vector multiplication _Linear Algebra review

1. Matrix * Vector : A x X = y

Method: each matrix row * vector
y (m,1)= A (mx n)*x (n,1)
y_i= sum (A_ij * x_j) over j.
to get yi, multiply A's i_th row with elements of vector x, and add them up.

Result dimension:
Row number = matrix A row number;
Column number = vector x column number 1.
Matrix A column number = vector X row number

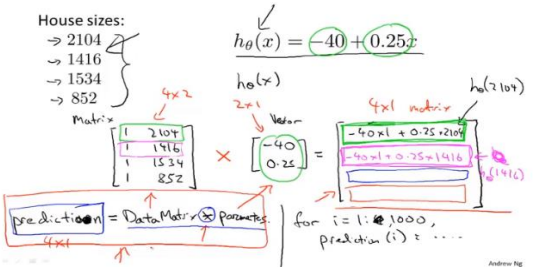
Details:



2. Example:

prediction house prize: with h function , for different house sizes:
use matrix * vector, could do using only one line code in Octave: prediciton = datamatrix x paramters;
while alternative method: for cycle:
for i=1:4:
prediction (i).
.. many cycles to run.

when have large numbers (e.g input det x) to deal with/predicit; using matrix not only one line of code, but also **much more computation efficient.**



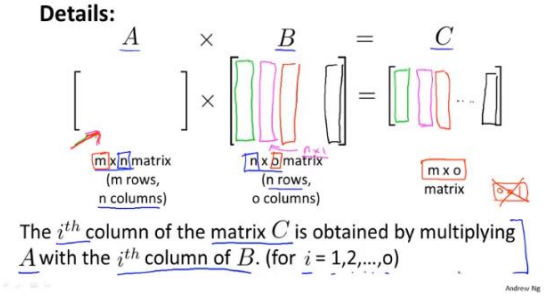
3_17. Matrix-matrix multiplication _Linear Algebra review

Matrix-matrix : A (m,n) X B (n, o) = C (m, o)

1. Dimension requirment:
A column number = B row number

2. Result dimenson:
C row number = A row number,
C column number = B column number
could take **matrix multipy o number vector B' (n,1), final result tacks all result vectors together.**
A. B = A. (b2, ...b_o) = (A. b1, A. b2, ...A. b_o)

Details:

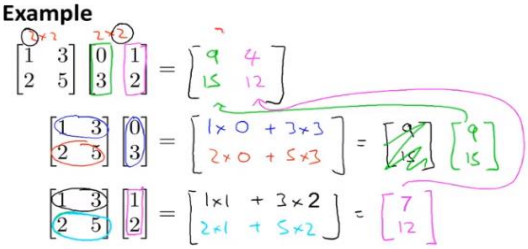


3. Method:

i_th columen of result matrix C, is obtained by muliplying matrix A with the i_th column of B.

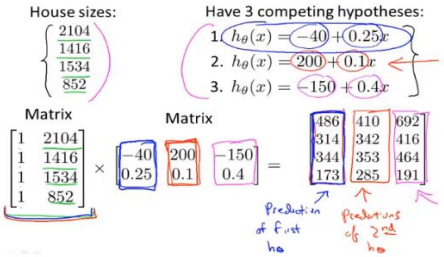
take first column of matrix B as vector and multiply matrix A, result is vector-first column of C;
take second column of matrix B as vector and multiply matrix , result vector is the second column of C;
And so on, last column of B, give last colume result of C.

Example



4. Example:
predict house price: with different hypotheses function h, for diffrent house size each h function paramter is one column of B.

-->One marices multiplication, get multiple prediction for diffrent house sizes with different predict function.
-->And, there are lost of good (open , free) linear algebra libraries in order to do this multiplicaiton step.

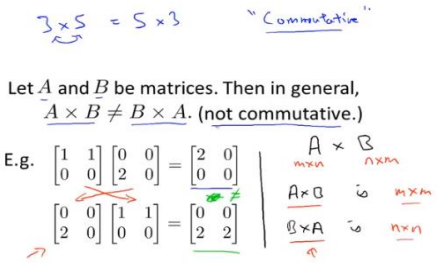


3_18. Matrix multiplication properties _Linear Algebra review

1: Not comutative: A X B /= B X A
comutative property: ordering of multiplication does not mater: A*B=B*A

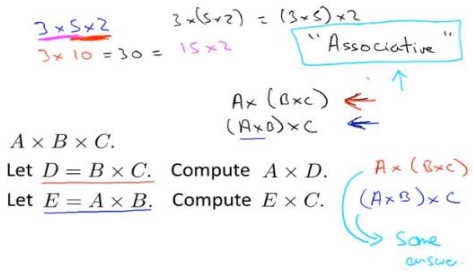
e.g:
- Dimension change:
  e. g: A (mxn) x B (n x m) =C (mxm), while B X A = C' (n x n)

-Resutl didffent even m=n,e.g right pic.



2. Is associative AX B X C= (A X B) X C = A X (B X C)

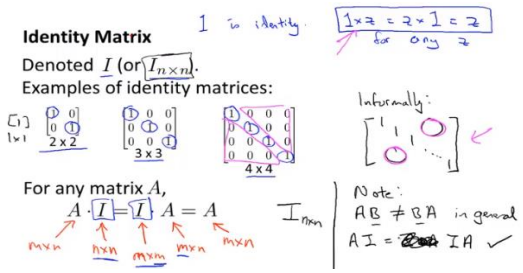
Associative property: calculation/multiplication sequence/priority does not mater



3. Identity Matrix: I or I_n x n

special matrix, simillar to real number 1: 1 x z = z x 1 =z

Identity matrix property:' comutative'
for any matrix A: A (m,n) X I (n,n) = I (m,m) X A (m,n) =A (m,n)

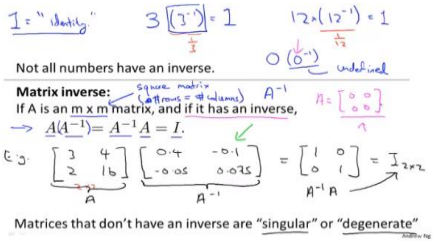


3_19. Matrix inverse and transpose _Linear Algebra review

1.Matrix Inverse
if A has an inverse: A * A-1 = A-1 * A = /

Sigular / Degenerate matrices: matrices that do not have an inverse.
like zeor matrix.

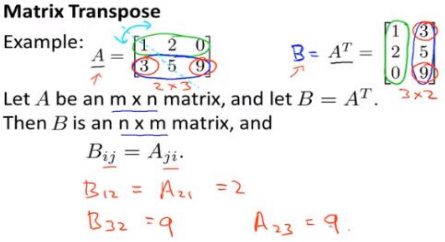
- Not all numbers have an in verse, not all matrix have an inverse;
- Only square matrix, have possbility to have inverse
- Many open source could use to compute matrix inverse
  e.g: In Octive, pinv (A) -->inverse A.



2. Matrix Transpose A (m,n),

Method:
chang each row to column positon; / take 45 degreee axis-go through first value, and then flipping matrix along this 45 degree axis

Transpose B=A_T
Bij = Aji.



W5-Linear Regression with multiple variables

5_27. Multiple features_Linear Regression with multiple variables

Multiple variable: have more than one features use to predict output y.

1. Notation: Multiple features

n=number of features
x(i)= input (feature vectpr) of i_th training example
x(i)_j = value of feature j in i_th training example

Multiple features (variables).

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

$m = 47$

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

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

2. Hypothesis:

sum of all features:
h_θ (x_i) =θ_0 + θ_1 *x(i)_1 + θ_2 * x(i)_2 + θ_3 * x(i)_3 + θ_4 * x(i)_4 + ... +θ_n * x(i)_n

for convinient take x(i)_0=1--->
X(i) =[x(i)_0, x(i)_1, ...x(i)_n)]T
θ=[θ_0, θ_1, θ_2, .. θ_n] T

h_θ (x_i)=θT *X

Hypothesis:

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

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

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

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

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

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

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

$= \theta^T x$

θ^T: $(n+1) \times 1$ matrix

Multivariate linear regression.

5_28. Gradient descent for multiple variables_Linear Regression with multiple variables

Gradient descent:
take paramter as vector θ=[θ_0, θ_1, θ_2, .. θ_n] T;
simultaneously update for evey feature paramter: θ_0, θ_1, θ_2, .. θ_n

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

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

Cost function:

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

($J(\theta)$)

Gradient descent:

Repeat {

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

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

}

Gradient descent : feature n=1 vs n>1

feature number n=1 ----> paramter θ_0, θ_1

feature number n>1 ---->paramter θ_0, θ_, .. θ_n:
derivative of θ_0, θ_1, formula same as n=1, (only now h function involve more features).
Same rule could apply to other parameter derivative.

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 $j = 0, \ldots, n$)

}

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

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

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

5_29. Gradient descent in practive: feature scaling_Linear Regression with multiple variables

1. Skewed ellipical shape cost function: feature scale differe large

E.g: only two features for examples, x1 (0-2000 range), x2 (1-5 range);
-->Cost function J(θ) is very skewed ellipical shape (2000 to 5 ratio)
-->if run gradient descent on this cost function, **may end up taking a long time and can oscillate back and forth and take a long time before it can finally find its way to the global minimum.**

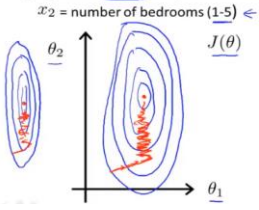
more skewed cost function, gradient is more harder taking it's way, meandering around, take a long time to find it's way to the global minimum.
(note: learning rate is same for every parameter, -->feature range differ, corresponding weights also differ
-->each weight gradient step: a * derivative:
when weights range differ big-->weight derivative differe big-->gradient step differe big-->some weight update towards global minimum, while others may overshooting-->total cost may decrease or increase : **cost oscillate**

note: if learning rate is unique for each weight -->no need feature scaling, will make cost decrease in every step.)

Feature Scaling

Idea: Make sure features are on a similar scale.

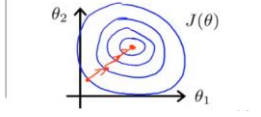
E.g. x_1 = size (0-2000 feet²)
x_2 = number of bedrooms (1-5)



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

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

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



2. Feature scaling: feature / value range

Idea: Make sure features are on a similar scale:

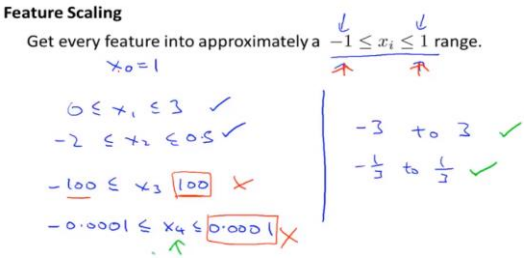
different feaure take on similar ranges of values, then gradient descent can converge more quickly.

By feature scaling, contur of cost function can become much less skewed, **conturs may looks like more circles.**
-->run gradient descent on cost function like this, could find **more direct patch to the global minimum (weight updating less overshooting: cost less oscillate with iteration steps),** rather than taking a much more convoluted path, trying to follow a much more complicated trajectory, to go the global minumum.

3. Feature scaling rule

Get every feature into **approximatedly a [-1<= x_i <= 1] range.**
E.g:
1< x (i)_1 < 3, this range is ok, close to [-1, 1];
-2<x(i)_2< 0.5, this range is ok, close to [-1,1];
-100 < x(i)_3 < 100, this range too big, no ok,
-0.0001 < x(i)_4 < 0.0001, this range too small, no ok

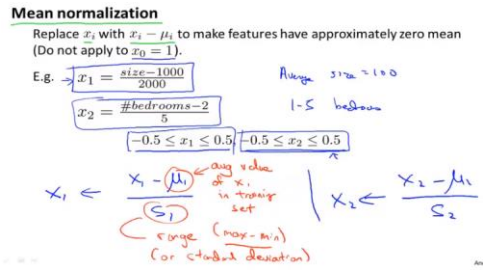
Note:
1. Normally below scale is ok: e.g : [-3, 3], [-1/3, 1/3];
2. Do not require all features are on exactly same range, but as long as they are all close enough to this range [-1, 1], **gradient descent** should work ok.
(note: scalling for gradient decent : -->weight updating more efficienly, leading cost to minimum)



4. Scaling method:_mean normalization
except divide feature range **size, could also use mean normalization**

method:
x(i)_j := x((i)_j-U_j) / s_j
replace x(i)_j with x(i)_j - u_j, to make features have approximately **zero mean (do not apply to x(i)_0=1)**

U_j: average of feature_j in the training sets : average of all i: x(i)_j. x(i): i example in training set.
S_j: range of feature j in the training set, max - min: max x(i)_j -min x(i')_j
Note: s_j could also use devation of j feature in trainin set. here max -min to get range is also ok.



Summary:
any value get feature into anything close to these sorts of ranges [-1,1] will be fine.
Feature scaling does not have to be extract, in order to get gradient descent to run quite a lot faster (and converge in a lot fewer iterations).

5_30. Gradient descent in practive II: learning rate_Linear Regression with multiple variables

1. 'Debuggling' : make sure gradient descent is working correctly

Plot draw: cost function after each iteration step in gradient descent(not pramter θ)

1.1. after certain steps of iteration, -->get new parmater θ-->calculate cost function J of θ.
1.2. Cost function should descent after every iterations.
give warning if gradient descent does not work correctly.

1.3. plot draw could help judge whether or not gradient descent has converged or not.
from this plot could find, after certain time steps, cost functioin does not go down much-like has flattened out here. Seems grandient descent kind of converged because cost function isn't going down much more.

Note:
1. converge steps for different applications could be big different.
2. difficult in advance predict how many steps needed to converge, usually using plot draw of cost function with iterations to see if converged or not.

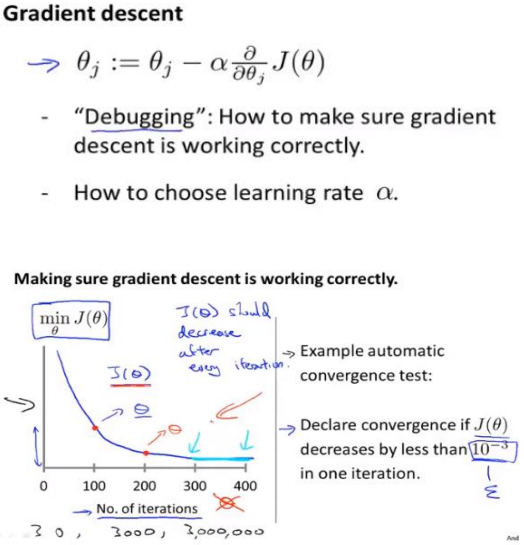
(note: plot draw of cost function with iterations:
-->1. debug: if gradient descent works;
-->2. check if converged or not
-->3. choosing learning rate
)

2. Automaticly Converge test

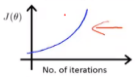
another method except plot draw to check if gradient descent converged or not.

Idea: declare convergence if cost function decreases by less than some small value ε in one iteration.

But choosing threshold ε is pretty difficult, prefer to look at plot-draw of cost function with iterations rather than rely on automaticly coverge test.

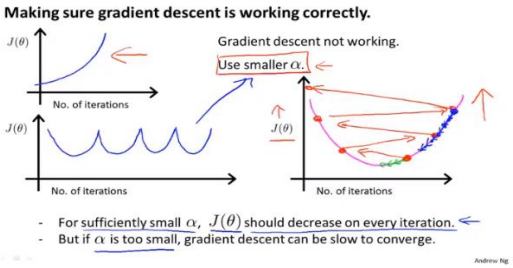


3. Uncorrect gradient descent



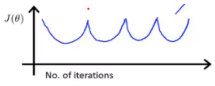
3.1: cost function single increase

If in plot-draw of cost funciton vs iterations, cost functions singal grow like above pic, then usually is learning rate too big:-->use smaller learning rate if no bug in coding
if learning rate too big, gradient descent may overshoot the minimum to a new postion and then overshoot again to another postion , keep on overshooting if learning rate too big, then end up getting worse and worse, getting higher value of cost function of θ.



3.2: cost function : go down for a while and then go up , then go down for a while and go up again..

-->use smaller learning rate also.



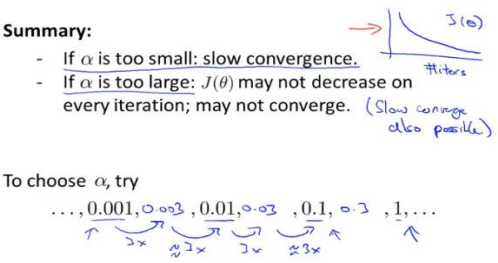
Note:
1. Mathmatically prooven, if learning rate is sufficiently small, cost function J decrease on every iteration.
So if gradient descient does not decrease, try smaller learning rate.

2. if learnig rate too small, gradient descent can be slow to converge.

Summary:
- if learning rate too small: slow convergence;
- if learning rate too large: cost function J may not decrease on every iteration; may not converge. Slow convergence also possible.
-->use plot draw of cost function with iterations, to figure out what's wrong.

To choose learning rate, try: ..., 0.001, 0.01, 0.1, 1, ...

Recommand:
run with different learning rates (raftly 3 times bigger of previous value, **till find the possible/reasonalbe largest value and smallest value**), plot cost funciton vs iterations, and choose one learning rate that seems to be **causing cost funciton decrease rapidly.**
(note: cost function decrease rapidly:
decrease: learning rate should be reasonably small
rapidly: learning rate should be reasonably large
)



5_31. Features and polynomial regression_Linear Regression with multiple variables

sometimes by **defining new features** , may get better model.

1. **Polynomial regression:** is the idea of choosing features

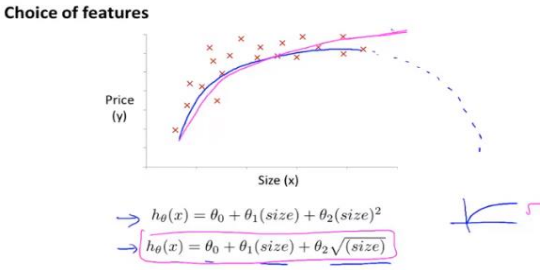
e.g: 1.1 Reduce feature number by creating new feature:

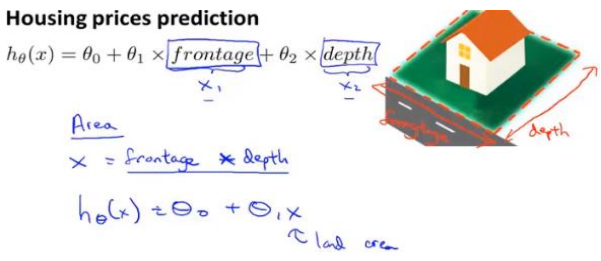
predict house prize: x1=width, x2=lengh h=θ_0 + θ_1* x1 + θ_2*x2
could create new feature x1 (area)= width * length---->h=θ_0 + θ_1* x1

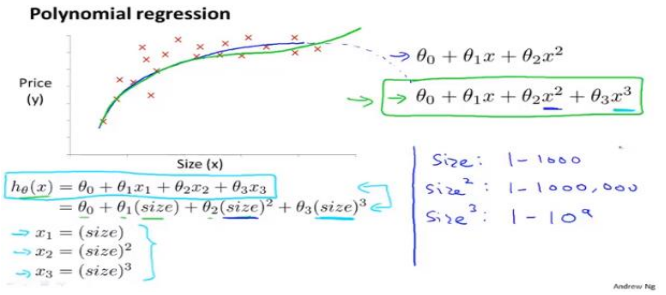
e.g 1.2: build linear regression by creating new feature

predict house size: x1=size, to better fit training data, h=θ_0 + θ_1* x1 + θ_2*x1^2 + θ_3*x1^3+...
creat new feature: x2=x1^2, x3=x1^3...--> h=θ_0 + θ_1* x1 + θ_2*x2 + θ_3*x3

Note: need to take care of feature scaling after creating new feature . in above example 2, feature x1, x2, x3 may has big range different, need to scaling





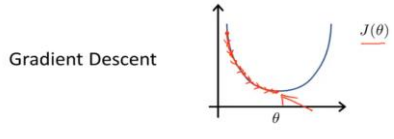


Summary:
1. Have choice what features to use;
2. By defining different features, can fit **more complex functions** to data

5_32. Normal equation_Linear Regression with multiple variables

1. **Normal equation: method to solve for parameter analytically**

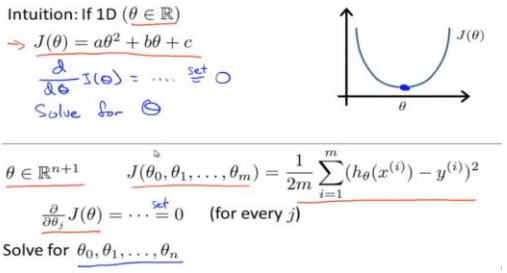
while gradient descent come to minimum by many gradient descent steps, normal equation solve optimal value for θ all at one go-in one step, get to the optimal value.



2. Normal equation intuition:

set partial derivative of cost function over each paramter =0, -->get paramter that minimize cost function J.

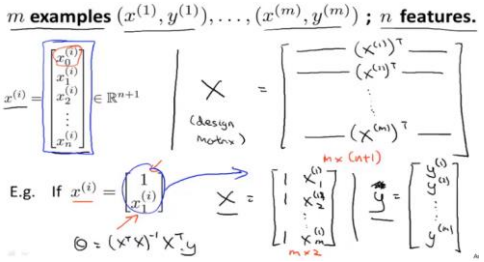
Note:
2.1 paramter dimension is **n+1** (n: input feature number)
2.1: need go ghrough all partial derivative of J over each paramter, and calculate seperately.
--> too long and too involved,

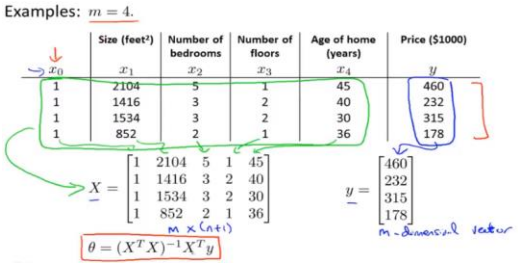


3. Normal equation calculation

θ= (X^T * X)$^{-1}$ * X^T * y

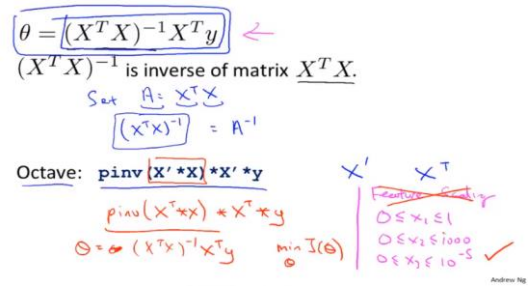
x(i) (n+1, 1)=[x(i)_0, x(i)_1, ...x(i)_n] ^{T}; X (m, n+1)=[x(1)^T; x(2)T; ...x(n)T]; y (m,1)=[y1, y2, y3, ..ym]T





4. Calcuation in Octave:

pinv : calcualte inverse
Note: feature scaling no necessary in normal equation, as feature scaling is for faser coverge in gradient descent



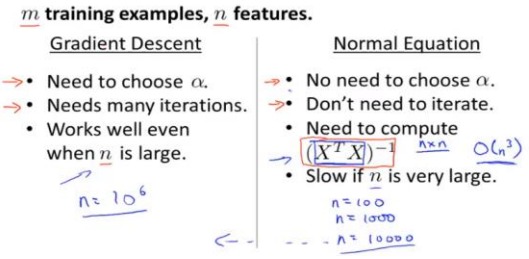
5. Comparision with gradient descent

m training examples, n features

Gradient descent:
- need to choose learning rate;
- needs many iteration;
- works well even when n is large; still reasonably efficient.

Normal equation:
- No need to choose learnig rate;
- Do not need to iterate;
- Need to compute (XT * X)-1: XT* X nxn dimension, computing time O(n^3)
- Slow if n is very large;

Note: if feature number n =100, 1000--->normal equation;
n=10000, may wonder which to use, may start to use gradient descent.



summary:
1. so long as the **number of features** is not too large, normal equiation gives us a greate alternative method to solve for the parameter θ.
So long as feature number is **less than 1000**, use normal equiation.
2. for mor complicated algorithm, like classification algorithm, line regression algorithm, normal equation actually do not work for those more sophisticated learning algorithm, have to resort to gradient descent.
logistic regression could use same equation, as cost J partial derivative is same as square error cost of linear algorithm.??
for this specific model of linear regression, normal equation can give a alternative that can be much faster than gradient descent.

5_33. Normal equation and non-invertibility_Linear Regression with multiple variables

XT * X: issue of being non-invertible should happen pretty rarely.

1. Method for non-invertible matrix XT*X
-Octave: pinv (X' *X): actually do right thing..
Pinv: pseudo inverse （伪逆）: could calculate the paramter θ wanted even if X'*X is non-invertible.
inv: inverse：逆

(note: XT*X: invertible-->r(X) = min(n, m)=n-->
1. m>n;
2. r(X) =n: feature x1, ..x_n 线性无关

X = (x(1), ...x(m)) = U .Λ. VT
>> U: nxn: 单位正交矩阵
>> Λ: nxm: 对角矩阵， 对角值为X 的特征值， 从大到小排序;
>> V: nxn: 单位正交矩阵

X.XT = U.Λ^2. UT: -->invertable-->Λ^2=Λ. ΛT : nxn : 对角矩阵， 特征值的平方， invertable -->
X n个特征值不为0.-->r(X) =n
)

Normal equation

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

- What if $X^T X$ is non-invertible? (singular/ degenerate)
- Octave: `pinv(X'*X)*X'*y`

2. Causes for non-invertible XT*X

2.1; Redundant features (linearly dependent)
e.g: x1= size in feet^2, x2=size in m^2.

2.2 Too many features (m<=n): try to find n+1 paramter to fit m example
- delete some features or use regulation (fit lot of pramters into small training data)

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

- Redundant features (linearly dependent).
  E.g. x_1 = size in feet^2
  x_2 = size in m^2
  $x_1 = (3.28)^2 x_2$
- Too many features (e.g. $m \leq n$).
  - Delete some features, or use regularization.

Summary: if XT * X is non-invertible:
1. check feature: if there is redundant features, delete it until no redundant features.: feature 线性无关
2. Then check if have too many features (m<n): delete features if bare to **use fewer features, or consider use regulation: n<m**

6_36. Basic operations_Octave Tutorial

4.6 Octave programming enviroment
use in MT programming;
Octave:free open source software.
Use Octave or Matlabe many learning algorithm become just a few lnes of code to implement.

Normal process:
1. First prototype learning algorithm/ software in Octave;
   Because software in Octave makes it incredibly fast to implement these learning algorithms.
2. Only after gotten it to work, then might migrate it to C++ or Java or whatever.

--->In this way much faster than if starting out in C++.

Matlab works well too, but too expensive.

Recommend to do excersize in this course in Octave

W7-Logistic Regression

7_44. Classificaiton_Logistic Regression

1. Linear regression for classification

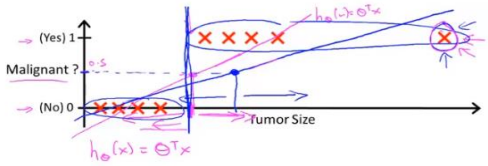
e.g:
h_θ(x)>=0.5, predict 'y=1';
h_θ(x)<0.5, predict 'y=0'.

when **training data is small and close, linear regression seems ok for clasifciation.**
but when trainign data is larger, like right pic. right point, linear regression changes to fit all data (for minimizing distance of prediction & training set), threshod x correspondg to h(x)=0.5 change to larger ->classification for smaller x does not right.

- Linear regression use for calssification task is not good idea.;
- Linear regression prediciton h_θ(x) can be >1 or <0, even training data is 0 or 1.

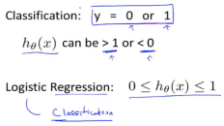
question: logistic regression decisonboundary is θT*x =t-->still linear boundary, how could fit right example better?:
-->linear boundary: as long as meet boundary spec, does not matter beyong how much.-->output is 1, cost is 0.
Linear regression: when used in classification problem, not only need to meeting the prediction line, but also the distance to prediction line is cost, need to be small.--->**prediciton line forced to be close to each training example.**
while logistic model ~~only meet linear boundary, cost is 0 cost is:~~ -lop(p)-->-θT.x >>0

-->different cost function/optimizing object-->learn different decision boundary
((eg.: take θ=(θ_0, θ_1))--->plot labled y vs x (only one feature x1)
>>1. logistic classification: optimizing object: θ_0+θ_1 *x1>>0 for y =1
>>. Linear regeression: optimizing object: θ_1*x_1 + θ_0: close to labeled y 1/0
-->get different decision boundary)



→ 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"

Logistic regression: 0<= h_θ(x) <=1



Summarize:

classification problem: **variable y is discrete value.**
Classificaiton : positive or negative definaiton does not matter that much.

Two-class problem: result only two, 1, or 0;
Multi-class problem: result more than 2.

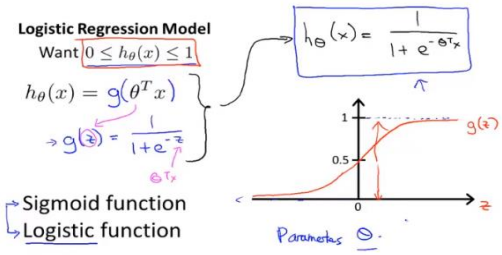
7_45. Hypothesis Representation_Logistic Regression

1. Logistic regression model: want 0<= h_θ(x) <=1

h_θ(x) =g(θT *x)

sigmoid function / logistic function: g(z)= 1/ (1+e^-z)

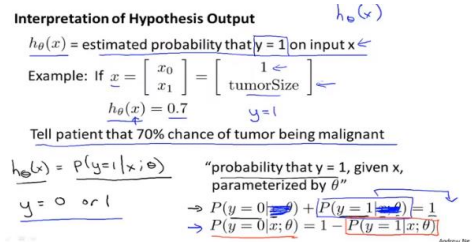
use this function, fit paramters θ to training data. Training data-->choose paramter-->use sigmoid/logistic function make predictions.



2. Interpretation of hypothesis Output

h_θ(x)= estimated probability that y=1 on input X
h_θ(x)=P(y=1 | x, θ) : 'probability that y=1, given x, paraeterized by θ'

as only have two result y=1, y=0, --> P(y=0 | x, θ) =1-P(y=1 | x, θ)

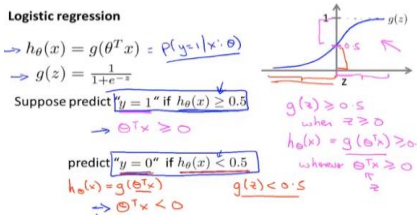


7_46. Decision boundary_Logistic Regression

1. Decision intuiton

h_θ(x) =g(θT *x)

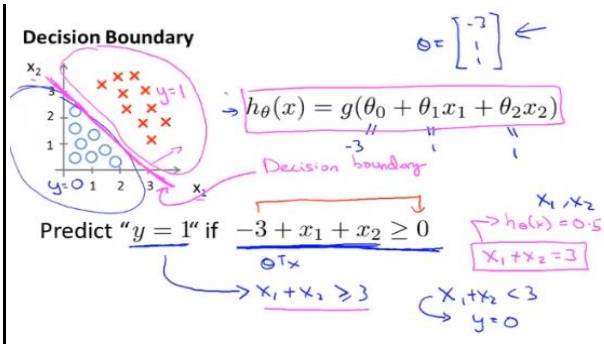
suppose predict :
y=1 if h_θ(x) >=0.5--- <----- θT *x >= 0
y=0 if h_θ(x) <0.5--- <----- θT *x < 0

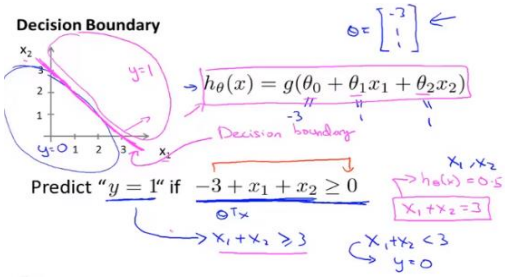


2. Decision Boundary_Linear

decison bounday: for chosen paramters θ: take θT *x = 0, this line is decsion boundary.

Note:
-Decision boundary is property of h_θ(x) =g(θT *x) and paramter, not training data.
once paramter is choosen, decision boundary is defined. trainig data is not used to define decison boundary, but may be used to fit the paramters θ





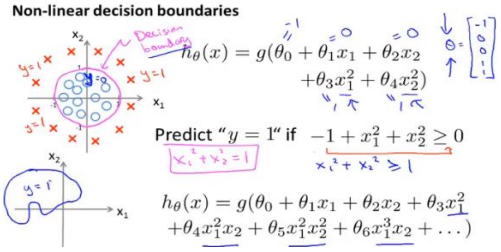
3. Decision Boundary_Non-linear

when have right pic. Training data, could:
add extra higher order polynomial terms to the features, Like In linear regression.

Once paramter is choosen, make take θT *x = 0;
this line is the decison bounday

Note:
-Decision boundary is property of h_θ(x) =g(θT *x) and paramter, not training data.
once paramter is choosen, decision boundary is defined. trainig data is not used to define decison boundary, but may be used to fit the paramters θ

Linear regression for classification: could not get good decison bounday.
maybe linear regression also get good decison boundary , by adding new feature??
(no: as optimizing object is different; linear regreassion forced decision boundary to be close to training set;
while logistic regression optiming object : is maximize probability)



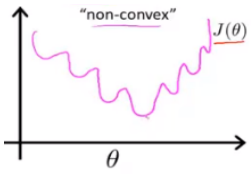
7_47. Cost function_Logistic Regression

Fit paramter for logistic regression; define cost function used to fit paramters.

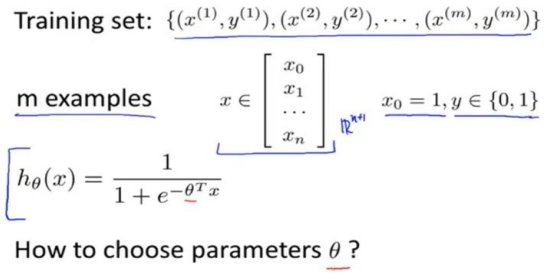
1. Cost function _linear cost function?

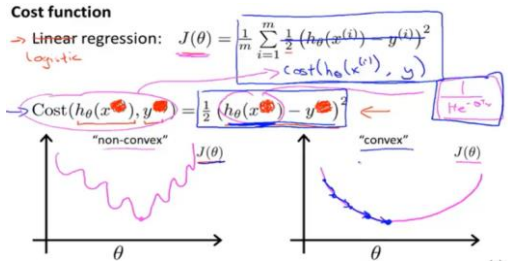
define cost (h_θ(x), y)=1/2 * (h_θ(x) - y)^2
Linear regression: J(θ)= 1/m (sum of cost (h_θ(x_i), y_i) over all examples)

cost (h_θ(x), y)=1/2 * (h_θ(x) - y)^2,
does not work well for logistic regression-->cost function J is non-convex (cause by intro. of sigmoid function), gradient descent could not gurantee convergence.
Need find another kind of cost function_ 'convex shape', so to gurantte gradient descent would converge to the global optima.
(note: cost fucntion purpose is to make graidient step/weight updating step bigger when prediction is differe big from label;
e.g: linear regression: square error : -->weight derivative dJ/dθ = (h_θ(x) - y)*x , will be bigger if h differe big from label y
if logistic regression, use square error: cost = (h_θ(x) - y)^2, -->dJ/dθ =(1-h)*h*(h-y)*x: when h-y big, derivative may not big)



Note: for one signal example
if use squared error-->dθ = (h-y)*h(1-h)*x: when prediction h is far from labled y, derivative θ is not big due to factor: h*(1-h)
if use log: l =- (ylog (h) +(1-y) * log (1-h)) -->dθ = (y-h)*x, when prediction h is far from labled y, derivative θ is also big.





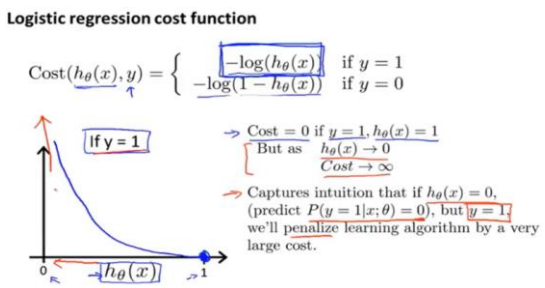
2. Cost function_Logistic regression

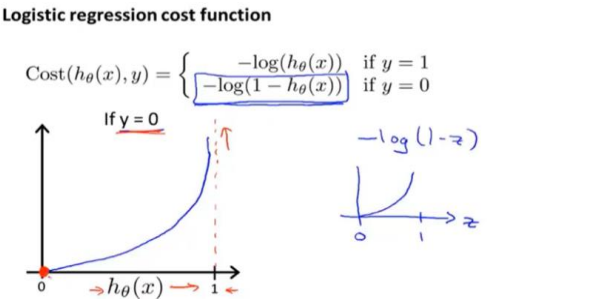
Cost (h, y)=
if y=1, cost =-log (h);(maximum probability)
if y=0, cost =-log(1-h);

2.1: if y=1 , cost= -log (h)
if y=1, h=1 --->cost=0:
h=1-->P(y^=1 | x, θ)=1, match label data 'y=1', cost =0.
if y=1, h=0, cost -->huge:
h=0-->P(y^=0 | x, θ)=1, doest not match label data 'y=1', cost =huge. penalize learnign lagorithm by a very large cost.

(note: cost function define:
object -->make derivative dJ/dθ =(h-y)*x
-->dJ/dh * dh/dz * dz/dθ = dJ/dh *h*(1-h)*x = (h-y)*x
-->dJ/dh = (h-y)/h/(1-h)
-->J = - (y*log(h) + (1-y)*log(1-h))

2.1: if y=0 , cost= -log (1-h)
if y=0, h=1 --->cost=huge:
h=1-->P(y^=1 | x, θ)=1, doest not match label data 'y=1', cost =huge. penalize learnign lagorithm by a very large cost.
if y=0, h=0, cost =0:
h=0-->P(y^=0 | x, θ)=1, match label data 'y=0', cost =0.





7_48. Simipifed cost function and gradient desent_Logistic Regression

Note: y=0 or 1 always (in training data)

1. Simplifed cost function

Purpose:
combine into one equation more convient to write out cost function and derive gradient descent

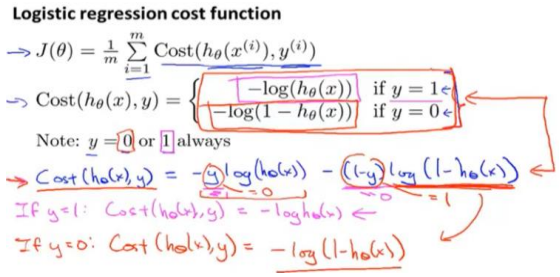
singal example cost:
cost_i= - y * log (h) - (1-y) * log (1-h)

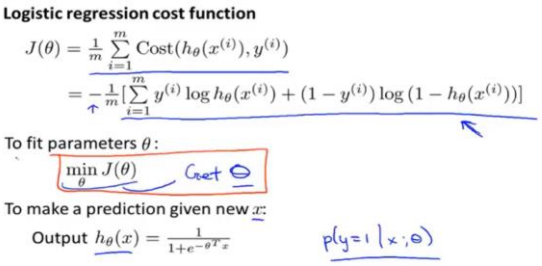
for each signle example i:
cost_i= -log (h) if y=1, =-log (1-h) if y=0;
the combined equation is same for singal example cost_i;

for all training set , cost
cost =sum (cost_i). since signal cost_i is same whether in combined equation or seperated equation-->overall cost also same in these two method.

Note:
- could use this combination , key point is traing data y is always is only 0 or 1.
- factor used here is 1/m, :
J=1/m (sum of cost_i)
for linear regression, cost_i=1/2* (h-y)^2;
for logistic regression, cost_i= y*log(h)-(1-y)*log (1-h)

- Use this function found paramter minimilizing cost functio J, then calcualte output h using choosen paramter. Output is the probability P(y=1|x, θ)





2. Reason for choosing this particular function

-This function can be derived from statistics （统计） using the principle of **maximum likelihood estimation** （最大似然）, which is the idea in statisc for how to efficiently find paramters for different models.
- This function is CONVEX

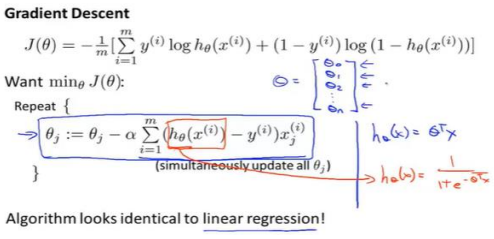
Note: another view of cost function, derived from likelihood estimation:
（后验概率： 已知y, 预测最可能的x 分布） max probability: P(y=1|x, θ) = p(y=1|x,θ)^y *(1- p(y=1|x,θ))^(1-y)
maximize probability log p= y* log(h) + (1-y) log (1-h)-->
minimize cost: J= - log (p)

3. Gradient descent

for each paramerter **θ_j**: θ_j:= θ_j - a * derivative of J over θ_j,
simultaneously update all θ_j.

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

Note:
- paramter gradient descent algorithm looks identical to linear regression.
Only different is : hypothesis funciton h. for linear regresion $h=\theta^T * X$, logistic regression $h=1/(1+e^{-\theta T} * X)$;
- could use vectorization to update all parameters in one time:
Gradient with parameter vector θ=[θ_0, ...θ_n+1]
θ:=θ-a * sum ((h(X)- Y).*X, x=1)
θ:=θ-a /m * X . (H-Y)T



4. Moniter Gradient descent

- same method as linear regresssion could apply on logistic regression to make sure converge correctly.
- feature scaling could also apply on logistic regession to make gradeint descent converge faster.

7_49. Advanced optimization_Logistic Regression

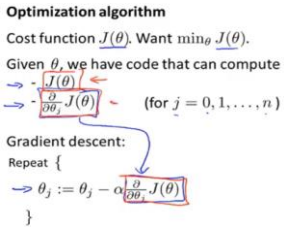
Purpose: to run much more quickly with gradient descent.

1. Optimization algorithm

Gradient descent:
given paramter, **have code that compute J, and derivative of J, than plug in parameter gradient descent, which can then try to minimize the cost function.**

Note:
for gradient descent, only compute derivative of J is ok;
for monitering cost function convergence, need compute cost fucntion J also.

(note: for batch gradient descent: compute cost J periodly as need to scan all example, computation cost is expensive;
while forsochastic: could compute cost on every iteration, and average over like 1000 examples , then plot the averaged cost;
mini-batch: could compute cost on every iteration and plot-also average over certain iterations??
)



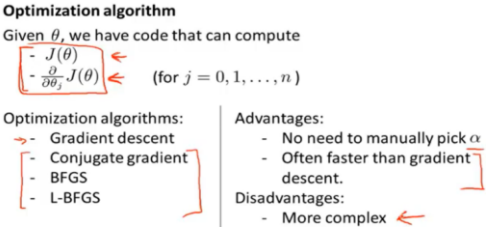
2. other sophisticated algorithm than gradient descent to minimize the cost function:

- Conjugate gradient;
- BFGS;
- L-BFGS

Advantages:
- No need to manually pick learning rate;
  Have a clever inner-loop-line search algorithm, automatically try out different values for the learning rate and automatically pick a good learning rate**. Even could pick different learning rate for each iteration.**

- Converge oftern faster than gradient descent.

Disadvantages:
- More complex



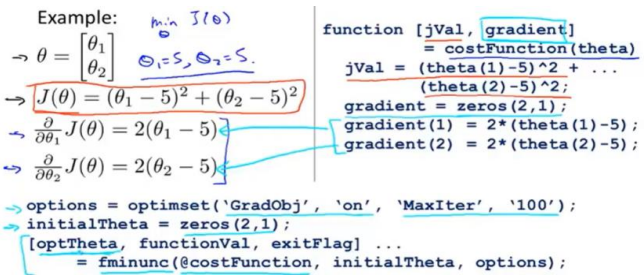
3. Example_other sophisticated algorithm

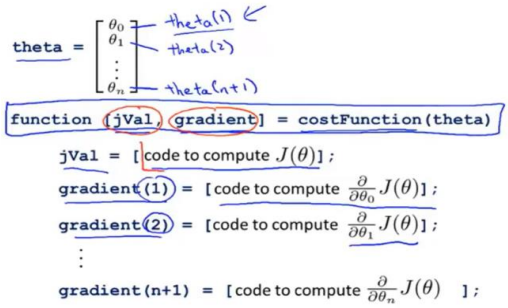
- Octave:
fminunc: function minimization unconstrained in Octave (无约束最小化函数）。

Note:
- paramter dimension have >=2 to use fminunc function;
- paramte vector start from θ_1, not θ_0.

- normally use when have large algorithm to run.

[optTheta, functionVal, existFlag]...= fminunc (@costFunction, initiaTheta, options);
feedback:
optTheta: paramtersl choosen;
functionVal: cost function value;
existFlag: converged or not; if is 1, converged.



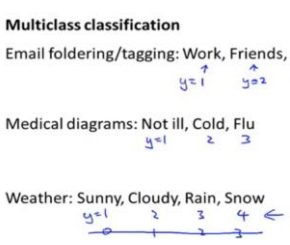


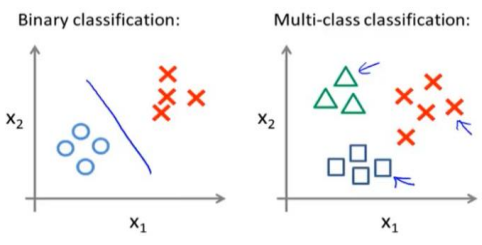
7_50.Multi-class classification: One-vs-all_Logistic Regression

e.g: output have more than two classfication

1. Training data exapmple:

for multi-class classficaiton, trainig data example like right pic.





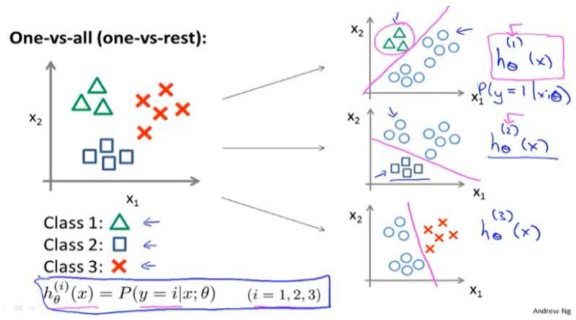
2. multi-classification Idea: one-versus-all/rest classification

one-versus-all/rest: for multi classificaiton:
take these training set into seperate binary classificaiton problems

- create a new training set: only one positive/class and negative output-all data except postive.
  -->get hyposthesis funciton h1 and decision boundary: p(y=1 |x, θ1)
  θ1: paramter vector for class 1.
- create new training set: another postive/class and negative output-->
  -->get hypothesis for this postive data: h2 and 2nd decison boundary: p(y=2 |x, θ2)

- Repeated, till find all the hypothesis h and decison boundary for all class. p(y=3 |x, θ3)

for given example X, run h(x) with all parameter for different class seperately, and choose the hightest possibility as output class.
(note: why not just use softmax..)



Summary:

- Train a logistic regression classifier h_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 (h_i (x)):
  run all classfiers h_i (x) on the input x, then pick the class i that maximizes among all the classes.

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

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

W8-Regularization

8_52.The problem of overfitting_Regularization

1. Example_Linear regression

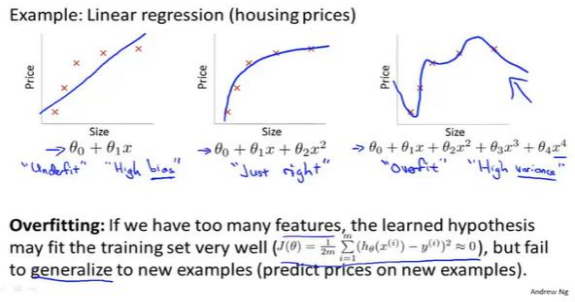
- **underfitting / high bias:** hypothesis model does not fit training data well
call bias: if fitting a straight line to the data, then as if the algorithm has a very strong preconception or very strong bias, that housing prices are going to vary linearly with their size. And despite the data to the contrary, depite the evidence of the contrary, its preconceptions still or bias still causes it to fit a straight line.

- **overift / high variance:** fitting such a high order polynomial （高阶多项式）almost as if it can fit almost any function, this face of possible hypothesis is just too large, **it's too variable. do not have enough data to contrain it, to give us a good hypothesis.**
(note: training set have too many features-->hypothesis high order polynomial, no enough constrain-training set,)

overfitting: if have too many features (high order polynomial), the learned hypothesis may fit the training set very well (cost funciton close to 0), but fail to generalize to new examples.

try too hard to fit training data, then fail to fit new samples.

Generalize: means how well a hpyothesis applies even to new examples-have not seen in training set.

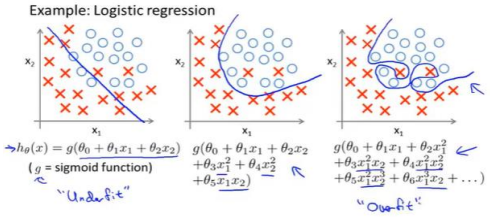


2. Example_Logistic regression

underfit / high bias: decison boundary does not fit training data well.

overfit / high variance: high order polynomial for decison boundary: generate high order polynomial terms of features.
algorithm try really hard to find a decsion boundary to fit training data, or go to great lengths to contort itself to fit every training data well.(make every examples' loss =0)

check decision boundary fitting with training data.



Summary:
check fitting status for both linear regression /logistic regression:

check fitting with training data:
Linear regression: fitting of hypothesis funciton prediciton with training data;
Logistic regression: fitting of decison boundary with training data.

3. Addressing

3.1: Reduce training set feature numbers:
- Manually select which features to keep.
  Keep important features.
- Model selection algorithm : model automatically decide which feature to keep. ??

Reducing feature number works well for overfitting.
But disadvantage: is by throwing away some features is also throwing away some of the information you have about the problem.
(note: same reason for using reduce_dimension on training set: do not use dimension reduciton unless have strong evidence current solution not working/too slow)

3.2 Regularization

- keep all features, but reduce magnitude / values of feature paramters
works well when **have a lot of features**, each of which contributes a bit to predicting y.

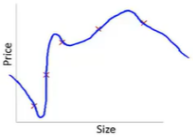
(note: root for overfitting: too many data features-->algorithm high polynomial, and no enough training set to constrain these parameters)

Addressing overfitting:

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.

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}



3.3: Plotting hypothesis(decision boundary)_does not work for addressing overfitting

when have only 1 or two features, we can just plot the hypothesis(decision boundary), and select the appropriate degree polynomial.

Plotting hypothesis could be one way try to decide what degree polynomial to use. but does not often work. more often we may have problems where we just have a lot of features, -->not just a matter selecting polynomial, and when have many features also becomes much harder to plot the data.**-->harder to visualize it to decide what features to keep or not.**

when have lots of feature and small training data, overfitting can be a problem.

(note: estimate different hypothesis on training & dev set: cost function J_train , J_dev of each type hypothesis vs iteration-->choose feature number:
1. just fit training set , no bias, no overfitting: J_train small
2. and generalize to dev set well: J_dev small)

8_53. Cost function_Regularization

1. Intuition:

Reduce feature weight by adding in cost function

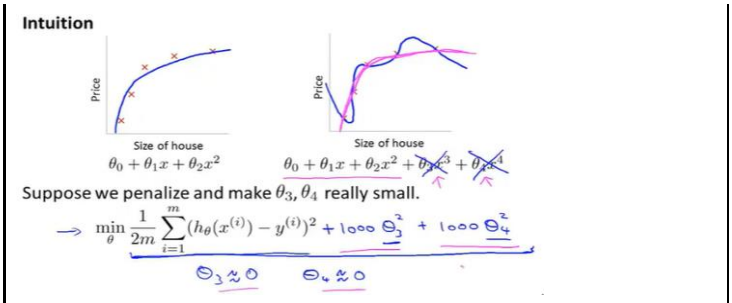
-->when weight is small engough, is kind of remove that corresponding feature.

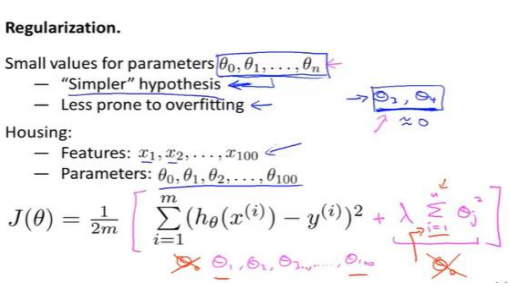
(note: penalting weight while considering label y value)

2. Regulation Idea:

small values for parameters θ_0, θ_1, θ_2..θ_n.----->
- 'Simpler ' hypothesis:
smaller parameters corresponds to usually smoother functions also simpler. also less prone to overfitting.
e.g: θ_3, 4,..n close to 0, then h nearly only have 2 features.

-Less prone to overfitting

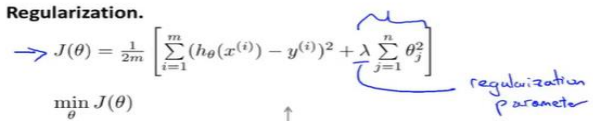


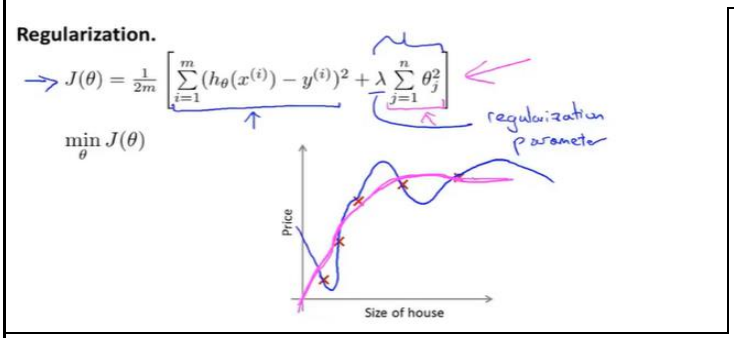


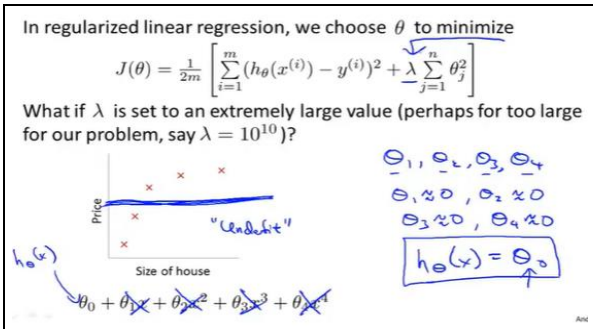
3. Regularizaiton:

as no idea which feature weight to shrink, modify cost function: adding regulation term in the end to shrink every single of parameter

Note:
shrink θ_1, θ_2..θ_n.
θ_0 not inclueded here, but very little influence on result.
(note: make dJ/dθ_k=0-->
θ_k = a *x_k /(λ+x_k^2) ; a: real value
1. λ =0:
feature x_k big: -->weight θ_k small
feature x_k very small-->weight big
2. λ !=0-->reduce weight optima







4. Regulation parameter λ:

Function: is a controls a trade off between two different goals.
first goal: without regulation term, would like to fit the training data well.
second goal: regulationterm: try to keep paramters small.

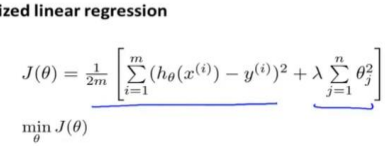
λ by balancing this two goal, to make hypothesis relatively simple to avoid overfitting.
e.g: when paramter is high, -->high order polynomial hypothesis, fitting traing data well, (first goal), yet regulation term is high-->reduce paramter, fitting error may increase, yet regulation term reduce.
(-->regulation will make first goal bigger: fit training set less well --> J_train increase, yet less overfitting)

need have good chosen for regulation parameter λ:
when λ is too big-->all paramter (excpet θ_0) close to 0 , hypothesis is a flat straight line, could not fit training set-->underfitting.

8_54. Regularized linear regression_Regularization

1. Object

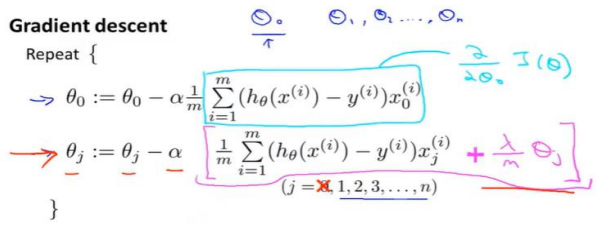
find paramter θ to minimizing cost function J.



2. Gradient descent_with regulation

2.1 Gradient descent formula

write gradient descent formula θ_0, θ_j seperately, as θ_0 is not involved in regulation term.

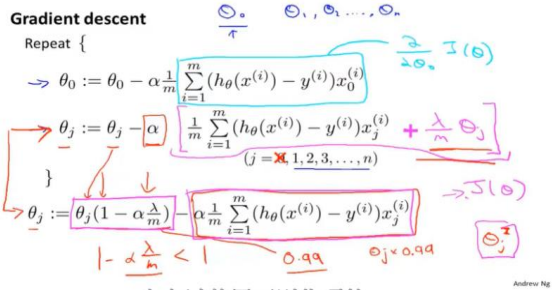


2.2: Interprenter of gradient descent

- First part: factor add for θ_j : 1-a*λ/m <1,
A little bit less than 1, shrinking the parameter a little bit.
(a is small, while m is very lage--> a little bit smaller than 1)

-Second part: exactly same as the formual befor add regulation term.

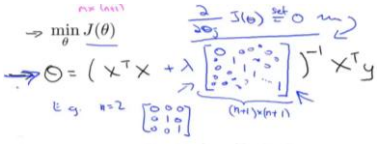
θ_j:=θ_j * (1-a*λ/m) - a/m * [sum (h_i-yi)*x_j, over i to all example m]



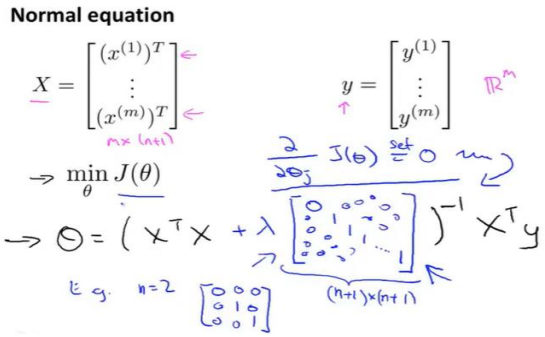
3. Normal equation_regulation

Method:
create new matrix X and labeled output vector y.-->find min J by making derative of J over each paramter =0

X=[x_i T, ...X_m T] T. mx (n+1)
x_i=[x_i (0), ...x_i(n+1)]



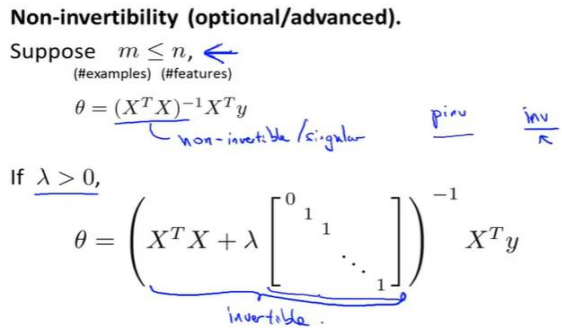
3.1: add new matrix (n+1, n+1) in the paramter:



3.2 non-invertibility

- without regulation:
when have fewer example than featurs m<=n, matrix XT *X is non-invertible / singular /degenerate.(pinv could calcuate fake invertible anyway)

- with regulation:
as long as regulation parameter λ is >0, XT*X + λ* new matrix is invertible.



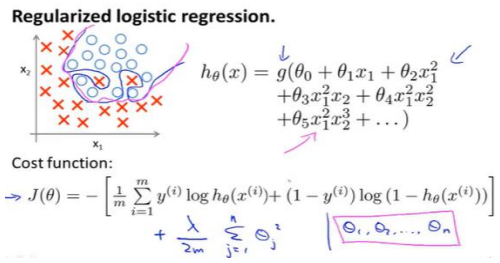
Summary:
adding regulation could fix overfitting when have lots of featus with small training set.

8_57. Regularized logistic regression_Regularization

1. Regularized logistic regression

method: add regulation term in cost function. Penalizing paramters when they are too big-->make decision boundary more reasonable.

E.g; when decison boundy is high order polynomial, -->overfitting.
By penalizing paramters in cost function with regulatation term,-->reduce paramter, -->make decision boundary more reasonable for seperating postive and negative examples.

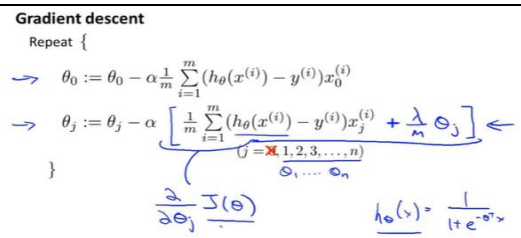


2. Gradient descent_with regulation

2.1 Gradient descent formula

write gradient descent formula θ_0, θ_j seperately, as θ_0 is not involved in regulation term.

Formula looks identical with linear regression, except only hypothesis function is different.



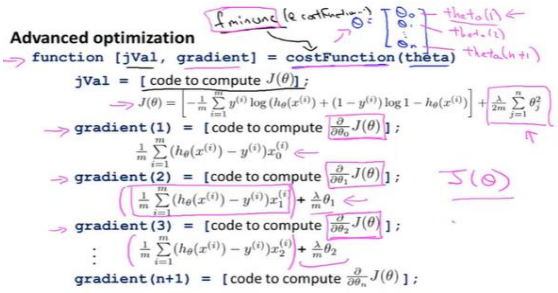
3. Advanced optimizationorithm

- need to define costFunction (θ) and gradient; then pass to fminunc

θ=[θ_0, ...θn]--->[θ_1, ...θ_n+1]

Note:
- paramter dimension have >=2 to use fminunc function;
- paramte vector start from θ_1, not θ_0.

in coding:
- update cost function jVal J with regulation term.
- update gradient of each paramter:
  gradient (1) is derivative of J over θ_0.
  gradient (2): update with regulation term over θ_1.



W9-Neural Networks: Representation

9_58. Non-linear hypotheses-Neural Networks: Representation

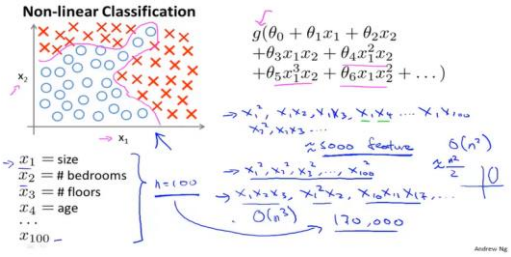
1. Machine learning _feature problem:
when feature number is large, feature number to consider

->>1. all quadritic features:
like 100, then the number of quadratic features grows （二次项的个数: x1*x2, x1*x3...）: rafly n^2/2

including all the quadratic features maybe not a good idea, may end up **overfitting and computation cos** is high for working with so many features.

- >>2. quadritic features:but only x1 ^2. x2^2. xn^2, only n quadratic features.
-->not enough features, **could not fit intersting hypothesis** (decision boundary simpler), can not fit the training set well

- >>3.quadritic features x1 ^2. x2^2. xn^2, and cubic / third order polynomial featuers: x1 *x2*x3, x1^2*x2....-->O(n^3)-->cubic features number about 170,000



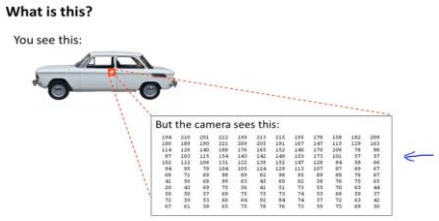
-->including these high order polynomial features when origianl features is large , will dramatically **blows up feature space** and is **not a good way to come up with additional features** with which to build non-linear classifiers when n is large.

yet for many machine learining problems, n is pretty large.
E.g in compute vison:

2. Example_Machine learning problem

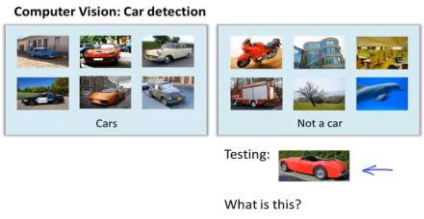
e.g: car classification

2.1. Car door handle classificaiton
Algorithm deal with below datas (pixl data , representing a cetain postion in pic.) to recognize this reprents a door handle



2.2 computer vision: car detection:

Normal process:
use machine learning to build a car detector: come up with a label training set-->learning algorithm trained a classifier-->test new image, see performance



2.3: understanding_why need non-linear hypothesis

> In the training set, pick a couple of pixel location in the image, -->plot the car at the location , at a certain point (depending on intensity of pixel one and pixel two),

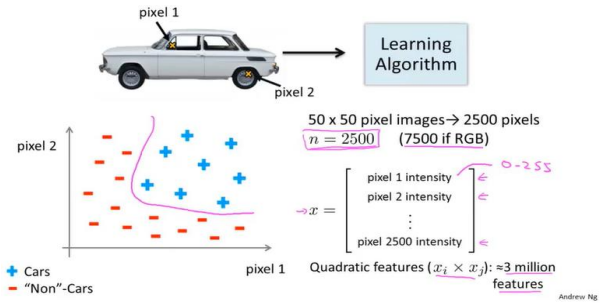
>Do same thing with other training set image: look at the same two pixel locations. that image have different intensity of pixel one and different intensity for pixel two,-->end up different location in the figure.

>plot some negative examples (not car)

>will find car and non car lying in different regions of the space.-->need non-linear hypothesis to seperate out the two calsses:

dimenson space:
- suppose image is 50 x 50 pixel-->2500 pixels , dimension of feature size will be n=2500 (7500 if RGB).

- if incuding all the quadratic features in learning algorithm, -->n^2/2 ~3 million features
too large to be reasonable. coumputation cost will be very expensive to find and represents all of these three million features per training example while feature vector is the list of all the pixel intensity.



Summary:
simple logistic regression, together with quadratic or the cubic features, is not a good way to learn complex nonlinear hypotheses, **when n is large**. Because **just end up with too many features.**

Neural network is a good way to learn complex nonlinear hypothesis , even when input features is large.

9_59. Neurons and the brain-Neural Networks: Representation

Neural network works well for different modern ML problem and AI.

1. Neural networks status:

Origins: algorithm that try to mimic the brain

Recent resurgence: state-of-the-art technique for many applications.
Reason: neural network computionly somewhat more expensive algorithm, it only because computes became fast enough, to really run large scale neural networks.

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

2. 'One learning algorithm' hypothesis

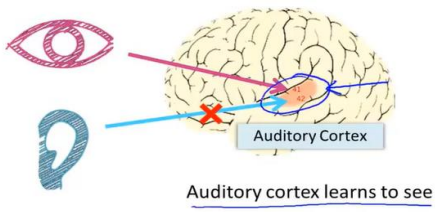
Hypothesis: brain do different things, is not worth like a thousand different programs, instead the brain does it is worth just a single learning algorithm

2.1 Proven for this hypothesis:-neuro-rewiring experiments
e.g 1: auditory cortex （听觉区）:

normal process:
ear pick up sound signal -->rute sound signal to auditory cortex -->understand words/vocie;

new process: cut wire from ear to auditory cortex, and rewire to eye
signal to the optic nerve, eventually gets routed to the auditory cortex**-->auditory cortex will learn to see (in every signle sense of the word see as we know: perform visual discrimination task: look at image and make decison by auditory cortex).**

The "one learning algorithm" hypothesis



Summary :
if the same piece of physical brain tissue can process sight, sound or touch, then maybe there is one learning algorithm that can process sight or sound or touch. inseatd of implement a thousand different programs/algorithms to do the thousand wonderful things that brain does.
maybe what we need to do is to find some approximation or to whatever the brain's learning algorithm is and implement that and let the brain **learned by itself: how to process these different types of data.**

3. Large extent of brain

seems we can plug in any sensor to any part of the brain, within the reason the brain will learn to deal with data.

More e.g:

- Seeing with tongue: help blind people to see.
Camara capture pic, pix-->transfer tonue by wire: each pixel tensity reflected by voltage-->learned to see

- Human echolocaton (sonar): help blind people to location
learn to interpret the pattern of sounds bouncing off enviroment.

- Haptic belt: Direction sense
similar with how birds can sense where north is.

- Implanting a 3rd eye: in frog,
frog will learn to use that eyse as well.

Sensor representations in the brain

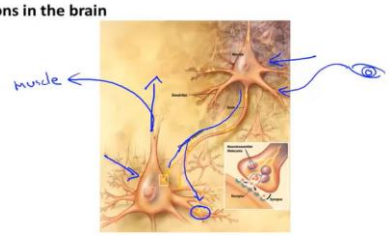
9_60. Model representation I- Neural networks: Representation

1. Neuron in the brain:

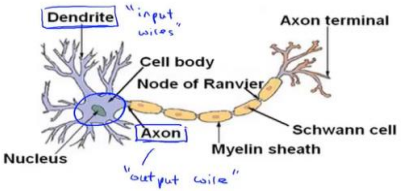
Simuliting neuros or networks of neurons in the brain.

1.1 Neuron unit:
input: Dendrite--->Nucleus --->Output Axon (computation), send message to other neurons (by sending little pulse of electricity), and will be input of next neurons.

Neurons in the brain



Neuron in the brain



2. Neuron model: losgistic unit (singal neuron)

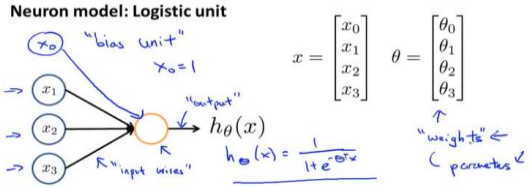
>Input: X (included x0 or not)
>Nucleus: computation
>output-->activation funciton

>parameter θ, also called weight.

Note:
x0 is added into X vector sometimes, as bias unite. X0=1;
h_θ(x)=1/ (1+e^-θT *x)

sigmoid (logistic) activation function:
an artificial neuron with a sigmoid or a logistic activaiton function. The activation function is another term for that non-linearality function.

Neuron model: Logistic unit



Sigmoid (logistic) activation function.
$g(z) = \frac{1}{1+e^{-z}}$

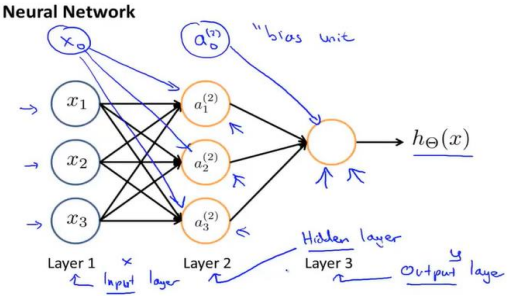
3. Neuron network

A group of different neurons strung together.
Final Output: h(x): hypothesis of input X.

Input layer: X;
Hidden layer: do not get to observe in the training set;
in supervised learnign, get to see input and correct output. whereas the **hidden layer are values do not get to observe in the training set;**
layers that are not input layer , output layer, all called hidden layer.
output layser: h(x)

Note:
>Input x0 could add or not.
>a0 could also add as an extra bias unite there: a0=1. (always is 1)

Neural Network

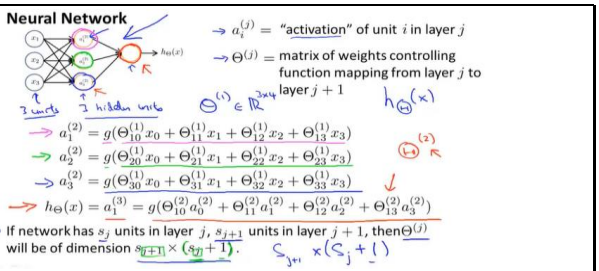


4. Neuron network_ computational step

Activaiton: a(j)_i
a(j)_i = 'activation' of unit i in layer j
by activation, means the value that is computed by and that is ouput by a specific.

Weight matrix:
Θ (j) =matrix of weights controling function mapping from layer j to layer j+1.

weight matrix dimension: Θ (j): **s_(j+1)** x **(s_j)+1** (bias)
feature number in next layer x feature number in current layer
s_j: units/feature number in layer j
s_(j+1): feature number in layer j+1.



9_61. Model representation II- Neural networks: Representation

How carry out computatin efficiently-->vectorization implementation
why neural network representation is a good idea and how they can help to learn complex non-linear hypotheses

1. Forward propagation: Vectorized implementation

1.1 Forward propagation:
Process of computing the activations, from the input, then the hidden, then the output layer.

called forward propagation as : start off with the activations of the input-units, then sort of forward-propagate that to the hidden layer, and compute the activations of the hidden layer. and then sort of forward-propagation that and compute the activations of the output layer.

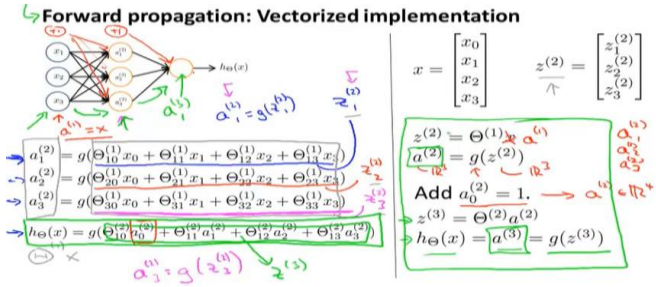
1.2: Vectorizing forward propagation:

x(i)=[x(i)_0, ..x(i)_n]
x(i)_0=1

- define extra term Z: for layer i
z_(i)= Θ_(i-1)T * a(i-1)
a_(i)=g(z(i)):
a & z have same dimension, g(z) apply sigmoid function to every element of Z.

- Take care of bias unit: add a_(i)_0=1

Vectorizing is a relatively efficient way of computing h of x.

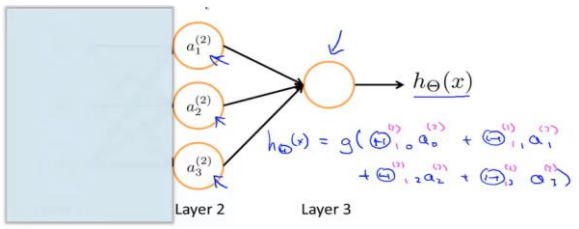


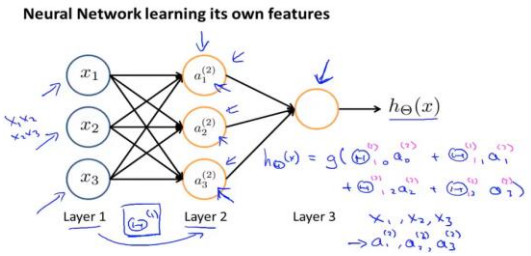
2. Neural network learning its own features

how could neural network help to learn intersting nonlinear hypothesis

'Standard logistic regression':
for each neuron unit: sort of logistic regression: using **logistic regression unite** to make a prediction h of x: h=g(ΘT*a)

but only the features fed into logistic regression, are these values computed by the hidden layer.





Summary:
Neural network is just like logistic regression, except that rather than using the original features x1, x2, ..xn, is **using these new features a1, a2, ...al.**
while a1, a2....al they themselves are **learned as functions of input.** Concretely the function mapping from layer 1 to layer 2 is determined by some other set of parmaters Θ_(1) (Θ_(1)_10, ...Θ_(1)_11, Θ_(1)_12,..Θ_(1)_20, ..)

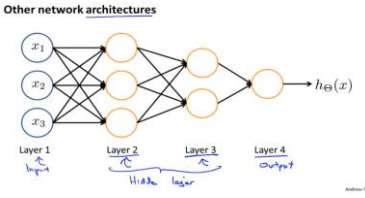
-->neural network, instead of being constrained to feed the orginal feature to logistic regression, it gets to learn its own features a(i) (a(i)_1, a(i)_2, ...) to feed into the logistic regression.
and **depending on what parameters it chooses for Θ_(1) can learn interesting and complex features.**and therefor can end up with a better hypothesis (better than considering only origianl features x1, x2.. or even polynomial features x1*x2..).

Neural network have flexibility to learn whatever feature want. and then feed into last unit: logistic regression for output.
(note: previous L-1 layer, learn new features based on original features, as input to the final logistic regression unit.)

3. Other network architectures
architecture: refers to how the different neurons are connected to each other.

Righ pic.: input feature feed into layer 1 to compute complex feartue and feed to layer 2 , compute new complex feature and feed to layer 3...till output.

hidden layer: anything not an input layer or output layer



Summary:

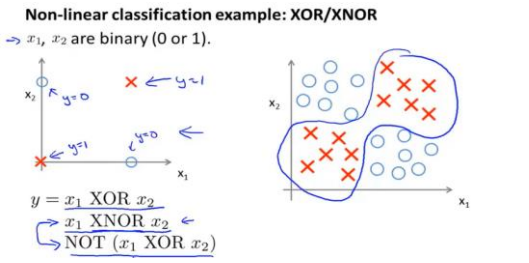
feed forward propagation in a neural network: start from the activations of the input layer, and forward propagate that to the first hidden layer, then the second layer, and then finally the output layer.
While in this process: vectorize computation.

9_63. Example and intuitions I- Neural networks: Representation

How neural netwrok compute complex features_logistic function

1. Non-linear classification example: XOR / XNOR

Function: output result of XOR inputs
x1, x2 are binary (0 or 1)--> function y=x1 XOR x2, X1 XNOR x2 (Not x1 XOR x2)



2. Simple example : AND function

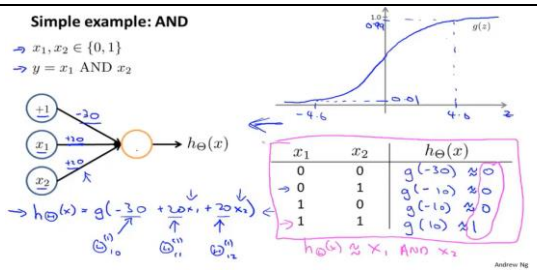
x1, x2 are bianry (0,1), x0=1;
y=x1 AND x2

-->neural network logistic regeression:
h(x)=g(θ_0* x0 + θ_1* x1 + θ_2* x2)

Note:
for sigmoid function, g(z) = 1/(1+1/e^z)), when Z=4.6-->g(z)=0.99~1; when z=-4.6, g(z)=0.01~0

by choosing paramter θ--->h(x) ~x1 AND x2: right pic.: θ=[-30; 20; 20]

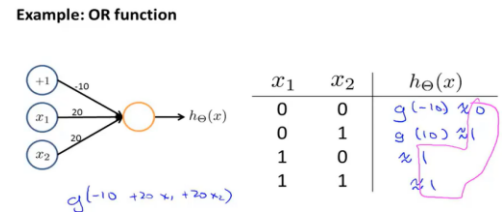
Right pic. pink table shows how neural network computes:



3. Eample: OR function

Similar as AND function, choose differnt paramter-->h(x) ~x1 OR x2

θ=[-10; 20; 20]



9_65. Example and intuitions II- Neural networks: Representation

How neural netwrok compute complex features

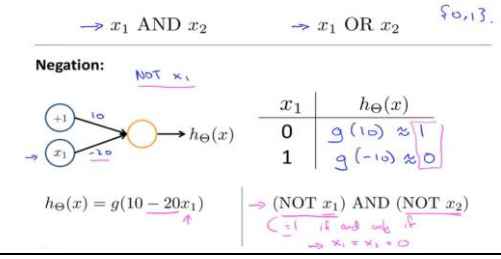
1. Negation_not x1

x1 is bianary(0,1)

Idea: put a large negative weight for variable want to negat
Paramter: θ=[10; -20]

2. Function (NOT x1) AND (NOT x2)

-->=1 only if x1=x2=0,
>could choose paramter Paramter: θ=[10; -20;-20]



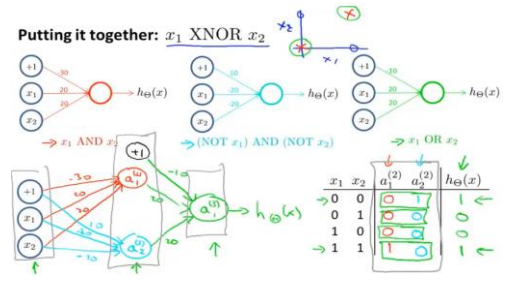
3.Putting it together: x1 XNOR x2

use two hidden layer:

- first hidden layer_2nd layer a(2):
a(2)_1~= x1 AND x2 function; binary (0,1)
a(2)_2 ~= (NOT x1) AND (NOT x2) function; binary (0,1)
a(2)_0=1

- 2nd hidden layer_2rd layr a(3)
a(3)_1 ~= x1 OR x2; binary (0,1)

(note: kind of breakdown problem-computing new feature , into computing many simple tasks/simple new features, whose computation based on previous new feature computed.)

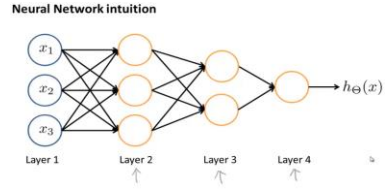


4. Neural network intuition:

hidden layer have slightly more complex functions.-->and yet add another layer, end up with an even more complex nonlinear function.<--why neural network can compute pretty complicated functions.

When have multi layers, have relative simple function of the inputs the second layers, but the 3rd layer can buildl on that and compute even more complex functions, and the layer after that can compute even more complex functions.

(note: deeper layer , compute more complex functions/features)



5. Example_Handwritten digit classificaiton

neural network application that captures above inuition of deeper layers computing more complex features.

Inpute: pic: handwritten character shown to network;
1rst hidden layer: different features: edges, and lines and so on detected.
right pic.-a visualization of the features computed by sort of the first layer of the network.
2nd hidden layer: harder to understand the visualization of this layer.
3rd hidden layer: even harder to understand;

-->probably have a hard time seeing what's going on much beyong the first hidden layer, but finally all the learned featurs get fed to the output layer, and show the final answer.

classifier work well: differnt type, noise , flipp, ...could look into video in this course

9_66. Multi-class classification- Neural networks: Representation

handwritten digig classificatiion is actually multi-call classficiation problem, have 10 possible categories.

1. Multi output units: One-vs-all method

try to recognize categaries of input: how many categaries in input image.
Category number of input: c

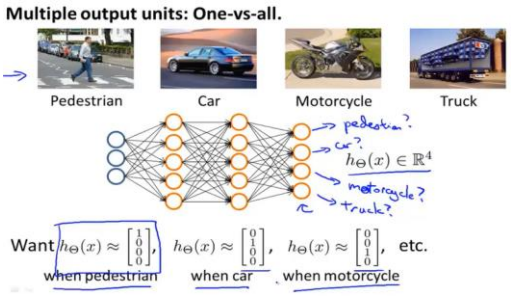
1.1 Output:
- neural network output vector dimension is (c,1): corresponding to classfication result of each catagory for one input image.

- output value expected h(x)~=[0,....1, 0, 1,..0] : 1 for catagories in image (may have multi categary 1 in image), 0 for catagories not in image.

e.g: right pic, have four catagories-->**have four logistic regression classfiers (a(4)_1,a(4)_2,a(4)_3, a(4)_4) , each of which is trying to recognize one of the four classes that we want to distinguish amongst.**

a(4)_i: reuslt value is (0-1), corresponding to the possible of being i catagory with given image.

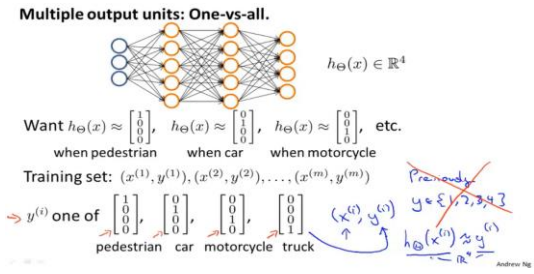
(note:
1. output: may have muliti object: -->multi classfire-logistic regression units in output layer
2. output: have only one object of multi possible class-->multil unit/classfire: softmax)



2. Training set representation

2.1 Training set:
pair: (x(1), y(1)), ...(x(m), y(m));
label y(i)=[0, ...1, 0.....0] (c x 1 dimension): depending on what the corresponding image x(i) is.

2.2 Object:
find h(x(i)) (c,1) ~ = y(i) (c, 1)
h(x) and label y have same dimention



W10-Neural networks: Learning

10_67. Cost function_Neural networks: Learning

Learning algorithm for fitting the parameters of the neural network given the training set.

(note: neural network: hard to find decision bundary: as the learned new feature: A[l-1]=f(x_1, ..x_n) to complicated to compute)

1. Neural Network (classification problem)

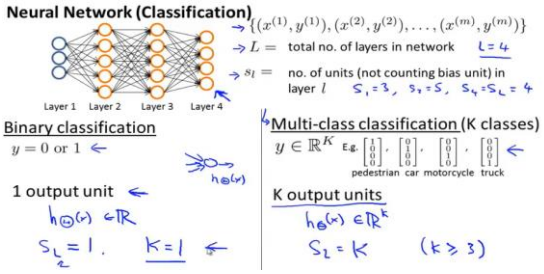
Training set:
pair: (x(1), y(1)), ...(x(m), y(m));
x(i) is.

K= class number
L= total layer number in network (including input and output layer)
s_l= units number in layer l, not counting bias unit (a(l)_0=1).

Output:
- **Binary classificaiton: y=0 / 1 (K<3)**
  - only one output unit (s_L / K=1), h(x) is an genger

- **Multi-class classificaiton (K classes, K>=3)**
- Y ~R^k
label y(i)=[0, ...1, 0.....0] (c x 1 dimension): depending on what the corresponding image

-K output units s_L=K, h(x) ~R^k

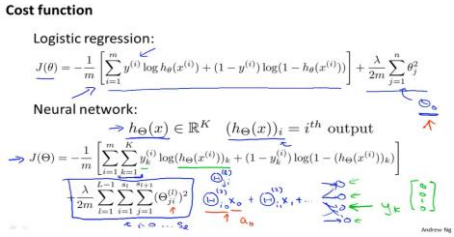


2. Cost function:

2.1 the generalization of the one used for logistic regression:
- θ_0 not included in regularizaiton
-first part: sum of all example standard cost without regularization;
- secodn part: regulation : sum of all paramters ^2 * λ/2/m

Logistic regression:

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



2.2 Neural network cost function:

one classfier cost function is same as standard logistic regression cost function-->
for all classfier cost function:
>1. first part:
>>> 1. sum of each classfiers' cost function: over all K classfier . for one example
>>>2 average of sum one exmple cost over all training example,

>2. second part: regulation: consider all paramters in neural net work: sum of each layer paramters' regulation. Θ_l (s_l+1, s_l): paramter matrix in layers l
Note: do not consider Θ0 eigher: Θ_l (j,i): j start from 1, to s_(l+1), i from 1 to s_l
Θ_l (j,0): multiply a_l(0) (=1)-->kind of like bias unit, same as logistic regression regularizaiton, will not sum over into regularization term (because we do not want to regularize them and string the values 0).
but if consider Θ_l (j,0) in regularization, it will work out same, does not make a big difference.

h(x): K dimensional output vector;
h(x)_j: selects out the i-th element of the vector h(x)
y(i): labeled output y for i_th example/input, K dimensional
y(i)_j: select out the j_th element of vector y(i)

10_68. Backpropagation algorithm_Neural networks: Learning

either use gradient computation or advanced algorithm, need code to compute cost funciton J and partial derivative of J over each paramter

1. Gradient computaiton
Partial derivation J over each paramter

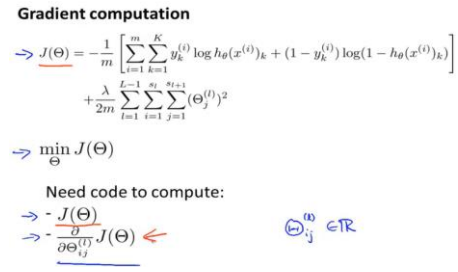
given one training example (x, y)

1.1 Forward computation : given the input, what the hypothesis actually output
a(1) = X;
Z(2) = Θ(1)^T*x
a(2) = g(Z(2)) (add a(2)_0)

Z(3) = Θ(2)^T*a(2)
a(3) = g(Z(3)) (add a(3)_0)

Z(4) = Θ(3)T*a(3)
a(4) = g(Z(4)) = h(X)_Θ

forward computiong by vectorlizing implementatin, allow to compute activation values for all of the neurons.



2. Backpropagtion algorithm_Gradient computation

a(l)_j: ativation of the j unit/neural duo in layer l
δ(l)_j: 'error' of node j in layer l: capture error in the activation of that neural do.

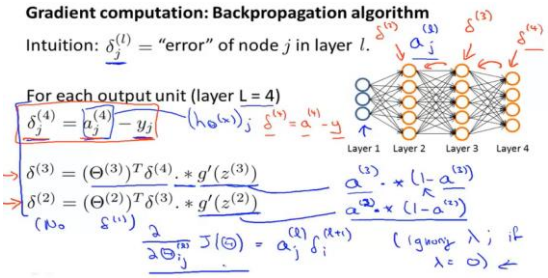
For each output unit
δ(L)_j = a(L)_j / h(x)_j - y_i
y_i = i_th element in the labled output vector

For all output unite: K dimension
δ(L) = a(L) / h(x) - y

Note:
>1. No δ(1): as first layer corresponding to input layer, and that's features observed in training set, so that does not have any error associated with that.- we do not really want to try to change those values.

>2. Name backpropagation comes from the fact:
Start to computing delta term for output layer, then go back a layer and compute the delta terms for this layer, and then go back another step to compute delta: backpropagating error of the output layer to previous layer.

>3. Using these delta term, could compute derivative terms for all of paramters.



公式推导

为了能展 **反向传播** 的魔法，我们首要要引入一个中间变量 δ ，定义为：

$$\delta_j^l = \frac{\partial J}{\partial z_j^l}$$

其中 l 为第几层， j 表示第 l 层的第几个神经元， z 为上次课程提到的中间变量（为了让式子看上去更清晰，**反向传播** 中的公式上标不使用括号）。 δ 被称为第 l 层第 j 个神经元的误差。
反向传播 就是先计算每一层每个神经元的误差，然后通过误差来得到梯度的。

首先来看输出层的误差：

$$\delta_j^L = \frac{\partial J}{\partial z_j^L}$$

对它使用 **链式法则** 得到：

$$\delta_j^L = \sum_{k=1}^{K} \frac{\partial J}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}$$

而只有当 $k == j$ 时，右边部分才不为 0 ，所以：

$$\delta_j^L = \frac{\partial J}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial J}{\partial a_j^L} g'(z_j^L) = a_j^L - y_j^L \qquad (1)$$

对于其它层的误差：

$$\delta_j^l = \frac{\partial J}{\partial z_j^l}$$

使用 **链式法则**：

$$\delta_j^l = \sum_{k=1}^{s_{l+1}} \frac{\partial J}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_{k=1}^{s_{l+1}} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

而其中：

$$z_k^{l+1} = \sum_{p=1}^{n_l} \Theta_{kp}^l g(z_p^l) + b_k^l$$

求偏导得：

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \Theta_{kj}^l g'(z_j^l)$$

所以：

$$\delta_j^l = \sum_{k=1}^{s_{l+1}} \Theta_{kj}^l \delta_k^{l+1} g'(z_j^l) = \sum_{k=1}^{s_{l+1}} \Theta_{kj}^l \delta_k^{l+1} a_j^l (1 - a_j^l) \qquad (2)$$

最后同样使用 **链式法则** 来计算：

$$\frac{\partial J}{\Theta_{ij}^l} = \sum_{k=1}^{s_{l+1}} \frac{\partial J}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial \Theta_{ij}^l} = \sum_{k=1}^{n_l} \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial \Theta_{ij}^l}$$

3. Backpropagtion algorithm_put all together

All training examples involved.

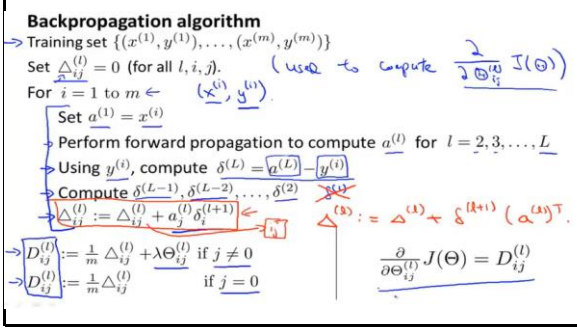
Δ(l))ij=0 (initalizing) (use to compute derivative of J over paramter)
(Δ(l))ij = dJ/dΘ(l)_ij)

process: look through training set.
>Loop : repeate through all training examples:

for one example (x(i(, y(i)):
>>>Set a(1)=x(i)
>>>Perform forward propgation to compute activationi value a(l)
>>>Using y(i), compute δ(L)=a(L)-y(i)
>>>>Compute δ(l): l~(2, L-1)
>>> compute error in whoel neural network for one training example:
Δ(l):=Δ(l) + δ(l+1) * a(l)T (accomulate these partial derivative terms): update of all values of i,j postion.

>out of loop

derivationve J over Θ(l)_ij:
D(l)_ij: = 1/m Δ(l)_ij + λ/m* Θ(l)_ij if j /=0;
D(l)_ij: = 1/m Δ(l)_ij if j =0 (corresponding to bias term)



由于：

$$z_k^{l+1} = \sum_{p=1}^{s_l} \Theta_{kp}^l g(z_p^l) + b_k^l$$

只有当 $k = i$ ， $p = j$ 时留下一项：

$$\frac{\partial J}{\Theta_{ij}^l} = g(z_j^l) \delta_i^{l+1} = a_j^l \delta_i^{l+1} \qquad (3)$$

反向传播

有了 (1) (2) (3) 式，就可以来完成 **反向传播** 算法了（需要注意的是刚才所推导的式子都是针对一组训练数据而言的）。

1. 对于所有的 l, i, j 初始化 $\Delta_{ij}^l = 0$
2. 对于 m 组训练数据， k 从 1 取到 m ：
· 令 $a^1 = x^{(k)}$
· 前向传播，计算各层激活向量 a^l
· 使用 (1) 式，计算输出层误差 δ^L
· 使用 (2) 式，计算其它层误差 $\delta^{L-1}, \delta^{L-2}, \ldots, \delta^2$
· 使用 (3) 式，累加 Δ_{ij}^l ， $\Delta_{ij}^l := \Delta_{ij}^l + a_j^l \delta_i^{l+1}$
3. 计算梯度矩阵：

$$D_{ij}^l = \begin{cases} \frac{1}{m} \Delta_{ij}^l + \frac{\lambda}{m} \Theta_{ij}^l & \text{if } j \neq 0 \\ \frac{1}{m} \Delta_{ij}^l & \text{if } j = 0 \end{cases}$$

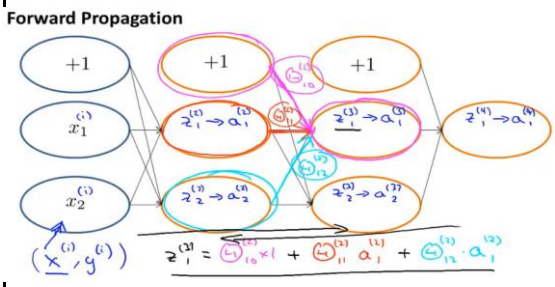
4. 更新权值 $\Theta^l := \Theta^l + \alpha D^l$

10_69. Backpropagation Intuition_Neural networks: Learning

1. Forward:

Input: (x(i), y(i))
x(i)-------> [z(1)-->a(1)] -------->[z(2)-->a(2)] -------->[z(3)-->a(3)] -------->[z(4)-->a(4)]

e.g:
z(3)_1 = Θ(2)_10 + Θ(2)_11 * a(2)_1 + Θ(2)_12 * a(2)_2
z(3)_2 = Θ(2)_20 + Θ(2)_21 * a(2)_1 + Θ(2)_22 * a(2)_2



2. What is backpropagation doing

single output cost function as right:

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

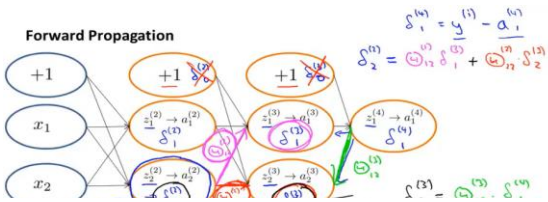
focusing on signel example, one output unit, no regulation.
could take it as square error, approximate way of mearsing distance of prediction and labled value.

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

3. Forward propagation:

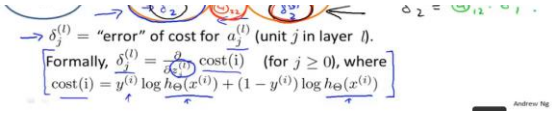
a(l)_j: ativation of the j unit/neural duo in layer l
δ(l)_j: 'error' of cost for a(l)_j (unit j in layer l)
'error' of node j in layer l: capture error in the activation of that neural duo.

δ(l)_j= partial derivative of cost function over z(l)_j:
if z(l)_j changes-->change output function -->end up change cost function.
Meausre: how much we want to change neural network weights, in order to affect the intermediate values (z) of the computation, so as to affect the final output that the neural network h (x), and therefore affect the overall cost.



Backward:
back progation is a lot like forward progation algorithm, but doing it backwards.
δ(4)_1=y(i)-a(4)_1
δ(2)_2=Θ(2)_12 * δ(3)_1 + Θ(2)_22 * δ(3)_2
bias δ(l)_0, not considered here (a[l]_0=1, has nothing to do with w[l-1], will not backprop to previous layer). biases backward included or not depending on backward algorithm and how to implement it. as bias always =1, no way to change it (could just compute it , but just scrad it-do not use it, because they will not end up being part needed to compute a derivative).

forward:
z(3)_1 = Θ(2)_10 + δ(2)_2* a(2)_1 + Θ(2)_12 * a(2)_2
z(3)_2 = Θ(2)_20 + Θ(2)_21 * a(2)_1 + Θ(2)_22 * a(2)_2



10_70. Implementation note: Unrolling parameters_Neural networks: Learning

Unroling matrix into vector, needed in order to use the advanced optimizatin routines.

1. Advanced optimization

>costFunction:
inpute parameter: **'theta', vector**
output: cost fucntion value 'jVal', and derivatives **'gradient'-vector**

>then pass this value to an advanced algorithm like fminucn
Note: fminunc is not the only one advanced algorithm, there are also other advanced authorization algorithms.
>>>take costFunction , and **intial paramterTheta (vector)**,

when using logistic regression, this works fine as theta and gradient is vector. but for a full neural network, we will have parameter matrix Θ(1), Θ(2), Θ(3), and gradient matrix D(1), D(2), D(3).

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

2. Unroll into vectors

thetaVec = [Theta1 (:); Theta2 (:); Theta3 (:)]
Dvec = [D1 (:); D2 (:); D3 (:)]

D / Theta (:): take all the element in matrix;
and unroll them and put all the elements into a big long vector thetaVec, Dvec.

3. Vector -->Matrix

Theta1 = reshape (thetaVec (1:110), 10, 11)

take vector thetaVec first 110 elements , and use reshape command to get 10x11 matrix.

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);
```

4. Learning algorithm

>have Initial paramter
>unroll to get 'initialThet' to pass to fminunc

Implement costFunction:
>get inpute thetaVec : thetal unrolled vector
>get back original matrix from thetaVec by using reshape function: **Θ1, Θ2, Θ3**
>Use forward prop/back prop to compute D1, D2, D3 and cost function J;
>Unroll D1, D2, D3 to get gradientVec, which can return by costFunction-a vector of these derivatives.

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`.

Summary:
matrix representation advantage: when parameters are **strored as matrices, it is more convenvinient when doing forward propagation and back propagation, and easier to take advantage of vecttorized implementatiosn.**

Vector representation advantage: when using advanced optimization algorithm, algorithms tend to assemue all paramters unrolled into a big long vector.

10_71. Gradient checking_Neural networks: Learning

Unfortunate property:
there are many ways to **have subtle bugs in back prop**, if run with gradient or some other algorithm, **it could actually looks like it's working**. Cost function J of theta may end up decreasing on evey iteraiton of gradient descent. (how this comes??)
Might just wind up with neural network that has a high level of error, than you would with a bug-free implementation. and you might just do not know it's subtle bugs that giving this performance.

Gradient check elimate all of these problem.
(note:
1. linear regression or logistic regressioin:
>>1. could check ploting of cost vs iterations-->debug (J_train: decrease with every iteration if learning rate is reasonable small
Out put = W*X/ sigmoid (W*X) : -->if gradient descent is wrong-->J_train plotting will be unnormal
2. Neural network:
>>. debug: by gradient check:
>> Output = g(w[l]* f(w[l-1) *f(.....)): -->cost =f(output): **d J/d parameter: is too complicated, if computation wrong, plotting J_train/dev may still decrease)**

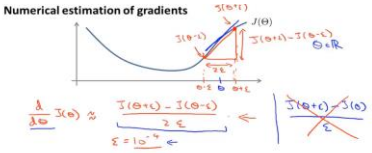
1.Numerical estimation of gradients

derivative of cost function **J over one specific parameter Θ (l)_ij-** right pic blue line, could check by slop of it's two side point Θ (l)_ij + /-ε

ε could take 10^-4, if too small may cause numrical problem.

Note: Two-sided differences gives a slightly more accurate estimate, rather than one-sided difference estimate.

Numerical estimation of gradients



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

2. Paramter vector (unrolled matrix into vector)

do two-sided difference for each paramter in parameter vector.

Implement in Octave: use 'for loop' for each parameter in vector.

>check two-sided difference calcuated above with Dvec (from back prop)
if very close , have more confience that computing derviatives correclty in back prop /other sophisticated algorithm.-->hope code run correcly and do a good job optimizing J of thelta.

Parameter vector θ

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

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

$\Theta_i + \varepsilon \to \Theta_i - \varepsilon$

$\frac{\partial}{\partial \Theta_i} J(\Theta)$

Check that `gradApprox` ≈ `DVec`

From back prop.

3. Implementation Note:

>Implement backprop to compute DVec (unrolled D1, D2, D3...)
>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 gradient checking code befor training classifier.
If run numerical gradient computation on every iteration of gradient descent, or in the inner loop of costFunction, code will be very slow as:

gradient checking is very computationally expensive, is a very slow way to try to approximate the derivative.. While backprop is much more computationally efficient way of computing the derivatives. So once checked backprop is doing right , should turn off gradient checking.

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.

10_72. Random initialization_Neural networks: Learning

when run gradient descent or other advanced algorithm, need pick some intial value for paramater thelta.
Then can slowly take steps to go downhill, using gradient descent, to minimize the function J of thelta.

1. Zero initializaiton:

Intialzing all paramters to 0 is ok for logistic regression. (note: error do not transfered by weight)

But does not work when traing neural network. (note: for layer 2- L-1: cost error transfered by weight:0)
E.g: right pic 3layer, two feature in fist lyaer, second layer, 1 feature in output
θ_ln_1=θ_ln_2=0
$X=[1; x1; x2]^T$

1.1 Forward
>Z(2)=θ_ln_1 ^{T}* X =0*X =0 vector
A(2)=[a(2)_1; a(2)_2]= g(Z(2)) !=0 (all unite values in layer 2 is same)
-->a(2)_1=a(2)_2 =...=a(2)_u2(u: unite number)
>Z(3)= θ_ln_2 ^{T}*A(2)=0*A(2)= 0 vector
A(3) = g(Z(3)) =g(0 vector) !=0 (all unite values in layer 3 is same)
-->a(3)_1=....=a(2)_u2(u: unite number)

>if θ_in_(l)=0
unites in each layer , have same value A(l)= g(θ_ln_(l) * A(l-1))= g(0 vector)
(generated by g_l(0 vector):
all Z(l) in each layer is 0 vector (due to paramter θ_ln_(l) -0), activation A(l) have same value in each element

>if θ_in_(l)=0, rest parameter in each layer θ_in_(l)=a_(l) (same value for parameters in same layer)

A(l)=a'_(l): all activation values in same layer are same.

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)` ?

1.2 Backprop

1rst iteration:
δ(L)=h(x)-y (!=0, != same value)
δ(L-1)=δ(L)*θ_in_(L-1)*g'(Z(L-1))
...
δ(2)=δ(3)*θ_in_(2)*g'(Z(2))

θ_update_(l)=θ_in_(l) - a* δ * A(l)

>if θ_in_(l)=0
-->δ(2)=δ(3)=...=δ(L-1)=0
-->θ_update_(l)=0 (paramters could not be updated if set all initial parmaters to 0)

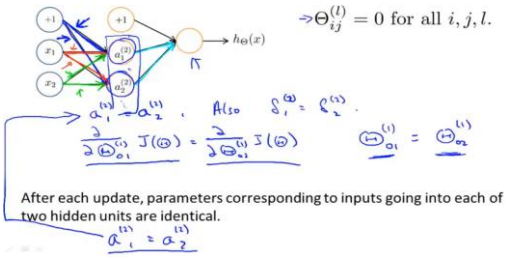
>if only 1 output (classfier, δ(L) is same in all uintes), θ_in_(1)=0, rest parameter in each layer θ_in_(l)=a_(l) (same value for parameters in same layer)
-->δ(l)=a'_(l),
θ_update_(l)=θ_in_(l) - a* δ * A(l)= same value for all unites in θ_update_(l)

>even after one gradient descent update, θ_update_(l) all values are still same
>>neural network really can not compute very intersting functions: all the hidden units in one layers are compting the exact same feature-this is highly redundant representation, as final logistic regression only gets to see one feature.

Note: for logistic regression:
only two layer : input X -->Z=W.X, A=Y
if all parameter =0, as δ(L)=h(x)-y (!=0, != same value)-->θ_update_(l)=θ_in_(l) - a* δ * A(l) = - a* δ * A(2)
paramter update ok, as δ(2) not =0
(note: but if mulit class output: multi logistic regression classfire: -->output layer multi unit:
then all output layer unit: have same weight value..-->only one classfier)

Zero initialization

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



from course:
δ(2)_1=δ(2)_2
>derivative of J over parameter θ_(1) _01= over parmater θ_(1)_02.
derivative of J over θ_(1) =δ(2) T* θ_(1)*g'(Z(1))
>updated after 1 iteration: θ_(1) _01=θ_(1) _02

after each update, paramters corresponding to inputs going into each two hidden unites are identical

2. Random initalization : symmetry breaking

Random intialization could solve above problem.

Intialize each θ_(l) _ij to a random value in [-ε, ε]:
>rand (10,11):
generate a random 10 x 11 dimensional matrix: all of the values are between 0 and 1.
>compute rand(10,11)*2 *ε-ε -->matrix values between [-ε, ε].

Note: ε here is different thing with ε used in grad checking

(note:
1. feature: 0 mean ans scaling to (-1,1) range, by (x-u)/s
s: could be max-min or varaince of feature x_k)
2. weight: intialize: [-ε, ε]:, close to 0)

Random initialization: Symmetry breaking

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

E.g.

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

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

Summary:
To train neural network, **randomly initialize the weigths to small values close to 0 between [-ε, ε]**, then do backprop and use eighter gradient descent or one of the advanced optimization algorithm to try to minimize cost function J of theta, as a function of the parameter theta.
Starting from just randomly chosen inital value for the parameters, and by **doing symmetry braking,** hopefuly gradient descent or advanced algorithm will be able to find a good value of theta.

(note: symmetry braking: needed for optimizing method that use gradient descent)

10_73. Puting it together_Neural networks: Learning

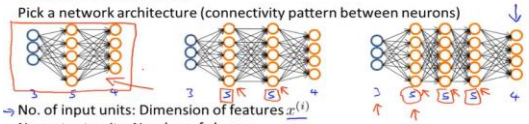
1. Training a neural network

pick a neural network architechture.
Architecture: connectivity pattern between the neuron. How many unites in hidden layer, and how many hidden layers.

>Features decided by training set:
>>input unites number: feature x(i) dimension;
>>Output unites number: number of class.
Note: labeled y for chosing classes from 1,2,3..10, , should rewirte y=[0,0,...1,0...]

Training a neural network

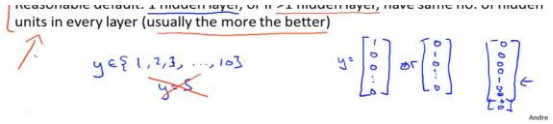
Pick a network architecture (connectivity pattern between neurons)



- 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

> Reasonable default: 1 hidden layer (perfectly reasonable default)
or if >1 hidden layer, have same number of hidden units in every layer (usually the more the better);
but if have a lot of hidden units, will be very computation expensive. Very often, having more hidden units is a good thing.
number of hidden layer units: maybe comparable to the dimension of input feature, or could be same number with input features or to maybe twice or three or four times of input feature number.

hidden units in each layer: inpute feature normally very large, -->still set hidden units to be comparable or times of input feature??

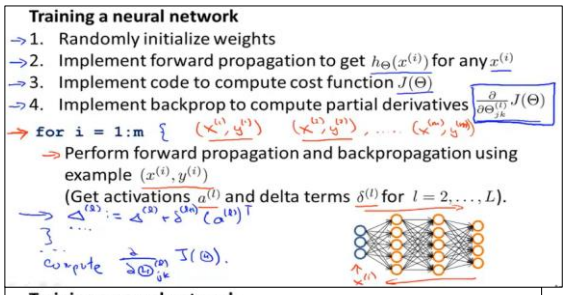


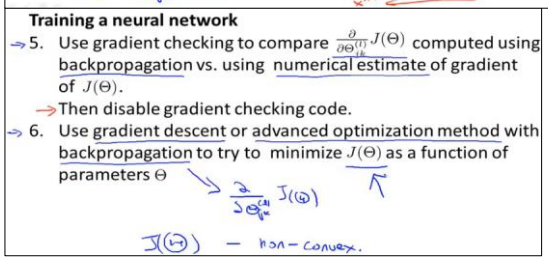
2. Training a neural network

> 1. Randomly intialize weights; to small values near zero.
>2. Implement forward progation to get output vector h(x) with given x.
>3. Implement code to compute cost function J of theta.
>4. Implement backprop to compute partial derivatives of J over θ_(l)_ij.
when do backprop, usually use for loop over the training example. there is advanced vectorization methods where don't have a for-loop over the m-training examples, but first time to implement backprop, should almost certainly be a for loop in the code when iterating over the examples :
x(1), y(1)---->forward prop -->backprop , then
x(2), y(2)---->forward prop -->backprop , then
...
till x(m), y(m)

first time try backprop better use for loop instead of advanced algorithm.

for i=1:m
{ Pefrom forward prop and back prop using example x(i), y(i):
get activations a(l) and delta terms δ(l): l=2....L
Δ(l):=Δ(l) + δ (l+1) * a(l)T
}
compute these partial drivative terms: d J(θ) /θ_(l)_ij:
also need to take regulation terms lambda as well.





>5. Use grad checking to compare d J(θ) /θ_(l)_ij computed using backprop vs using numerial estimate of gradient of J(θ).
Reassure backprop computation is correct , then disable gradient checking code.

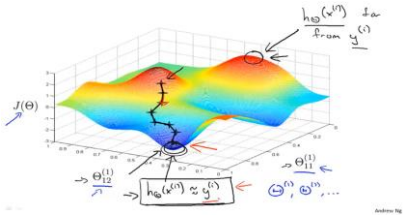
>6. Use gradient descent or advanced optimization method with backprop (for calculating partial derivatives d J(θ) /θ_(l)_ij)) to try to minimize J(θ) as a function over paramters θ.

Note:
for nueral netowrk , cost function J(θ) is non-convex. It can theoretically be susceptible to local minima. In fact gradient descent or other advanced algorithm can get stuck in local optima, but it turns out usually it is not a huge problem, as usuallly algorithm like gradient descent will do a great job minimizing cost function J of theta, and get a very good local minimum.

3. Intuition of backprop in neural network:

when implement backprop and use gradient descent or one of the advanced optimization methods, right pic shows what the algorightm is doing:
start from some initial point , and repeatedly go downhill. **Backprop is for computing the direction of the gradient**, and gradient descent is for taking little steps downhill, until hopefully it gets to a good local optima.

try to find the values of the paramters where the output values in the neural network closely matches the values of the labled y(i) (minizing cost funciton J): **low values of the cost function corresponds where J of theta is low and therefor corresponds to where the neural network happens to be fitting training set well. h(x_i)~=y(i) is needed to be true in order for J of theta to be small.**



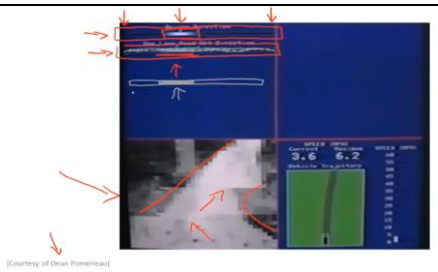
10_74. Autonomous driving example_Neural networks: Learning

Autonomous driving: getting a car to learn to drive itself.

1. Visualization:

> down in the lower left: **view seen by car, k**ind of see a road.
>on top in the left:
>> First horizental bar: d**irection selected by human driver** and is the location of this bright white band: left end-steering hard to the left; right end side-sterring hard to the right.
White band is a little bit to the left: means human driver at this point is steering slithly to the left.

>> Second horizental bar: **direction selected by learning algorithm**. white band means direction selected is a just slightly to the left.
before algorithm starts learning intially, the output is like a uniform grey band-corresponding to neural network having been randomly initialized, and intially having no idea how to drive the car, /which direction to select. and only after it's learned for a while and then to output like a solid white band in just a small part of the region correspinding to a particual steering direction.



2. Automous naviation experiment:

ALVINN: a system of artifical nerual networks, that learns to steer by watching a person drive.
NAVLABII: car used for this experiment

neural network setup: Learn to steer by watching a person drive.
Device: Sensore , computer, and actuators

> Intial step in configuring this car: training a network to steer
>>during training , human driver drive the car while car watches.
>>Once every two seconds, ALVINN igitizes a video image of the road ahead, and records the person's steering direction.
training image is reduced in resoution to 30 x 32 pixels.and provided as input to ALVINN's three-layer network.
>>using backprop learnig algorithm, ALVINN is trained to use same steering direction as the human driver for that image.

>>Intially steering directy is random, after about 2 minuts training, the network leans to accurately imitat the steering reactions of the human driver.

>>Same training procedure used in other road type

>After training: neural network to drive

>>12 times per second, ALVIINN digitizes an image, and feeds it to its neural networks,

>>each neural network, work in parallel , produces a steering direction and **a measure of its confidence in its response. steering direction from the most confident network.**
e.g: now the one-lane algorighm is used to control the vehicle.
when approach intersection, confidence of the one-lane network decreases.-->
after went through intersection, and the two-lane road ahead comes into view, the confidence of the two-lane network rises, then two-lane algorithm is selected to steer vechile on the two-lane road.

W11-Advice for applying Machine learning

10_77. Deciding what to try next_Advice for applying Machine learning

Purpose:
While improve algorithm, what are the **proxy avenues to try** next.

1. Debugging a learning algorithm

Problem: hypothesis on new set , have large error in its predictions.-->improve algorithm

Method:
>Get **more training examples**;
Not always help

>Try **smaller set of features**
(very effective for overfitting, yet will throw away useful info., use regulartition instead.)

>Try **getting additioanl features:**
e.g for predting house price: get more info. about the pieces of land .and so on.
>>before doing above things, helpful to know in advance if this is going to help.

>Try adding polynomial fatures (x1^2, x2^2, x1*x2, etc)
(help underfitting, make algorithm more powerful with more weights)

>Try decreasing regulation parameter **λ**

>Try increasing regulation parameter λ

each above method can scale to a long time project: spend may about 6 month on one term

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
- 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

Techni easily to rule out many of these options.

2. Machine learning diagnostic

Diagnostic: A test that 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.
when developing learning algorithms, as they will save you spending many months pursuing an avenue that coud have found out much earlier what not going to be fruitful.

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.

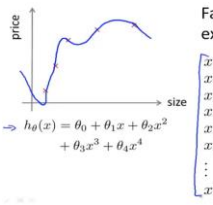
10_78. Evaluating a hypothesis_Advice for applying Machine learning

1. Evaluating hypothesis

Principle: how algorithm trained have low error in training set **, generalize to new examples not in training set.**

1.1 Method:
>Ploting hypothesis: to see overfit or not
very hard to plot when have many features.

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}

Andrew Ng

1.2 Standard process:

>Split datat set into : training set (70%)-m_traing, test set (30%)-m_test

Note: if there is any ordinary to the data, better split data randomly. Or **randomly reorder the exmaples** befor spliting into training set and test set.

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

30% Test set → $(x_{test}^{(1)}, y_{test}^{(1)}), (x_{test}^{(2)}, y_{test}^{(2)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = no. of test example

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

Andrew Ng

2. Training / test procedure for linear regression

>: Learn paramter θ from training data (minimizing training error J) 70% data

>Compute test set error (on 30% data), could use cost function
take parameters learned from first step and plug in here and compute test set error.

>>>**Error defination:** for linear regression, could use cost as error: sum squared error over all test examples.

(note:
1. cost function: is used as optimizing object , to get weight more close to labled y
2. Error: could be anything judging output good or not: could use
>>2.1 cost function measure distance of prediction-lable
>>2.2 : accuracy, recall, precision,..F1 etc.
)

Training/testing procedure for linear regression

- Learn parameter θ from training data (minimizing training error $J(\theta)$) 70%
- Compute test set error:

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

3. Training / test procedure for logistic regression

same procedure as liner regression. only cost function defination is different.

>: Learn paramter θ from training data (minimizing training error J) 70% data

>Compute test set error (on 30% data)
take parameters learned from first step and plug in here and compute test set error.

>>>>Error defination for logistic regression:
>Misclassification error (0/1 misclassificaiton error):
Alternative test sets metric:
0/1: get an example right /wrong.

e.g: hypothesis mislabeled the example:
>>>error (h(x), y)=1:
if h(x)>=0.5, y=0;
or if (h(x)<0.5, y=1

>>>error (h(x), y)=0: other cases.

Test error could define as: Test error =1/m_test * sum (error (h(x_i), y_i) over m_test data
why not use cost as error for logistic regression ?? (cost of logistic regression: is maximing probablity to optiming weight-->understanding/'visualization' not good as accuracy here;
)

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):

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

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

10_79. Model selection and training / validation /test sets_Advice for applying Machine learning

Model selection problems:
decide what **degree of polynomial / features** (neurla network: hidden layer number, hidden layer units number)to include to fit to a data set, or **regulation paramter λ**.

Overfitting example

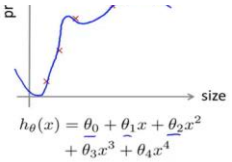
price

Once parameters $\theta_0, \theta_1, \ldots, \theta_4$

1. Overfitting example

hypothesis parameters trained in training set, does well in training set does not mean how well hypothesis (or decision boundary-decided by parameters) will generalize to new examples not seen in the training set.

General principle:
Once parameter well filts to some set of data, then the error of hypothesis is measured on that same data set, that's unlikely to be a good estimate of actual generalization error. (how well hypothesis generalize to new samples).



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.

2. Model selection (Linear regression, logistic classification)

2.1 New paramter: polynomial degree d.
chose polynomial order /degree to fit data.: d

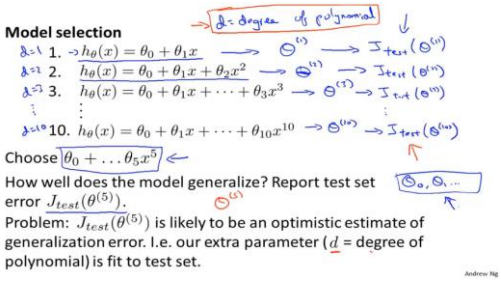
except parameter θ , have one more paramter d trying to determine using data set.

2.2. Process:
> choose one degree model;
> fit that model :
on training set, get paramter θ_d_i
> get some estimate of how well fitted hypothesis was generalize to new examples (dev set)..
compute test error.

take hypothesis corresponding to each trained ploynomial degree model, and measure performance on the test set.

>choose model (polynomial degree) has the lowest test set error.

note:
could d combine into hyperamteter metrics, and search hyperameters to use (random choice-not grid, corase to fine method..).



2.3: choosed model generalization performance
One way: look at the test error this choosen polynomial hypothesis had done in test set (which used to choose polynomial degree).

Yet is not a fair way to estimate how well hypothesis generalizes, as we've **fit this paramter d using test set**-chose the d value that give the best possible performance on the test set.
-->performance of paramter θ (for the model d chosed) on the test set, that's likely to be an overly optimisic estimate.
hypothes is likely to do better on this test set than it would on new exampels it hasn't seen before.

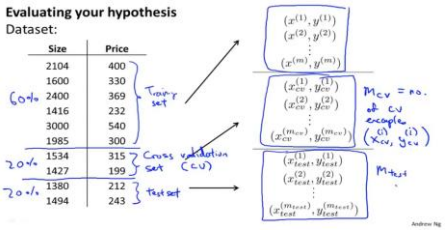
test set and dev set are same distribution, -->dev set feature shoud same as test set, -->using dev error as test error still a problem?-->algorithm hyparameter fitted on dev set, -->evalutation final choosed algorithm on test set.

3. Evaluation hypothesis

Dataset:
>Traing set: 60% data--m_training

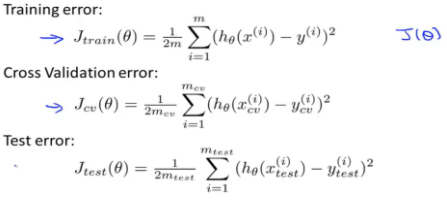
>Cross validation set (cv/v set): 20% data--m_cv

>Test set : 20% data--m_test



4. Train / validation / test error

cost function on different set: training error, valiation error and test error.



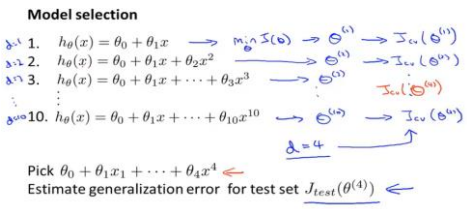
5. Model selection:

use valiation set instead of test set to select model (polynomial degree)

> for each potential choosen polynomail hypothesis di:
train on training set, Min J(θ)--->θ_trained---->compute J_cv (θ_train_di)

> pick model of the lowest J_cv (θ_train_d): fit polynomial degree using valiation set.

>Estimate generalization error for test set J_test (θ_train_d)
(note: generalization error: trained & choosed final algorithm performance on new examples not seen before)



Note:
many people report test error of hypothesis acutally on dev/test set (no dev set). If have massive test set, maybe not a terrible thing to do (less overfitting dev set).
Better do not do so and practice to have seperate train , validation and test sets.

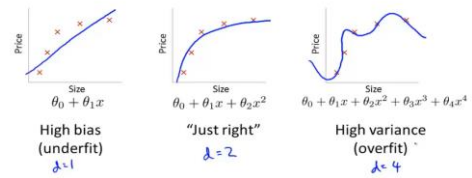
10_80. Diagnosting bias vs variance_Advice for applying Machine learning

If algorithm does not doing well, almost all the time it will be because have either a high bias problem, or a high variance problem: either underfitting or overfitting

1. Bias / variance intuition (from linear regression model /logistic classification model)

d: polynomial degree

High bias: underfit (when d is small)
High variance: overfit (when d is big)



2. Bias / variance defination

Train / validation / test error:
cost function on different set: training set, valiation set and test set.

2.1 Training and dev error vs polynomial degree

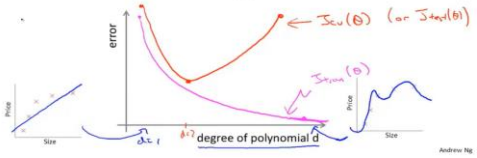
d is small-->underfitting: J_train is big, J_cv is big;
d is just right -->just right: J_train is small, J_cv is small
d is big --->overfitting: J_train is smaller, J_cv is big.

J_train will decrease as d increase: fitting training set better with higher polynomial hypothesis.
J_cv will be high when d is too small or big: not able to fit well when overfitting or underfitting.
J_test: similar with J_cv

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 $J_{test}(\theta)$)



3. Diagnosing bias vs varaince

if **algorithm performing less well: J_cv/ J_test is high**-->d is too small or big

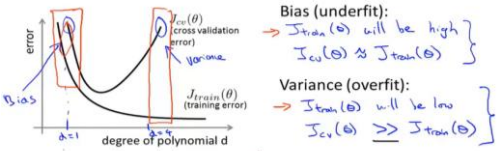
>High bias: d is too small:
J_train is also high; J_cv is maybe slightly higher than J_train;

>High variance: d is too big:
J_train is low: fitting training set very well;
J_cv is much bigger >> than J_train

(note: when dignosing, J_train, J_cv: regularization item not involved in calculation)

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?



10_81. Regularization and bias / variance_Advice for applying Machine learning

For preventing , regulation added to keep the values of the parameters small.

1. Linear regression with regularizaiton

regularization parameter: λ

for choosen model (polynomial degree d):
> High bias (underfit):
large λ-->all parameters θ are heavily penalized, --->small parameter (θ close to 0)
--->hypothesis end up more or less a flat, constant straight line, close to θ_0 (do not included in regulation)

this hypothesis do not fitting well data set.

>High variance (overfit)
Small λ-->largel parameter
if λ close to 0, given that we are fitting a high order polynomial, this is a usual overfitting setting.
given a high order polynomial, basically without regularization or very minimal regularization, end up with high variance overfitting setting.

Question: if λ=0, given that we are fiiting a just right /low order polynomial-->result in underfitting?

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) → $\lambda = 0$

2. Choosing the regularization parameter

Training / valiation / test error: J_train, J_cv, J_test
cost function without regularizaiton term on different set: training set, valiation set and test set. Note: for these three error calculaiton, Without regularizaiton term;

Cost function: J (θ, λ): consider regulization term.

Note:
cost function used to finding paramter to make hypethesis function h fit training set well, need to involve regularization into computation while training algorithm-purpose: penalting weights.-->increase J_train

while training /dev/test error is estimating trained hypethesis performance.
for linear regression, error also used squared error over all test examples, but without regularization, to reflect real error.

Choosing the regularization parameter λ

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

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

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

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

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

J_{train}, J_{cv}, J_{test}

2.1 Selecting process:

>1. Try different λ for cost function J (θ, λ): start from 0, then 0.001 (then may step up in multiples of two until some larger value)

>>for each λ value: similar process with choosing polynomial d:
>>>min cost function J(θ) with regularization term on training set --->θ_train_λi
>>> Compute validation error J_cv(θ_train_λi) on dev set .
Note: in validation error , regulization term not considred.

>>>pick the lowest J_cv(θ_train_λ)-->hypothesis function h(θ_train_λ)

>Report test error: generalize selected hypothesis function to test set which hypotheis function did not see before..

similar to polynomial degree selecting, using training set, dev set and test set.

(note:
training set: -->using optimizing object, find weights, minimize cost J
dev set: -->choose hyparameter , based on algorithm dev error
test set :--> evaluate final fixed algorithm performance/test error.)

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 $\lambda = 0$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01$ → $\min_\theta J(\theta)$ → $\theta^{(2)}$ → $J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$ → $\theta^{(3)}$ → $J_{cv}(\theta^{(3)})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$ → $\theta^{(5)}$ → $J_{cv}(\theta^{(5)})$
⋮
12. Try $\lambda = 10$ → $\theta^{(12)}$ → $J_{cv}(\theta^{(12)})$

10.24

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

3. Bias / Variance as a function of the regularization parameter λ

how training error and dev error vary as vary the regulaization parameter lambda λ

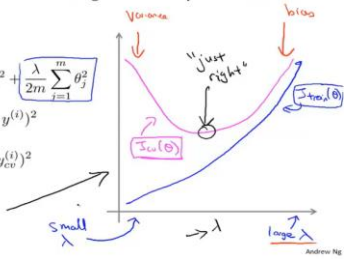
original cost function: J_o_ (θ, λ): consider regulization term.
Training error and validation error: J_train_(θ), J_cv_(θ): without considering regulariton term

Bias/variance as a function of the regularization parameter λ

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

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

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



> 1. λ is small: not using much regularization
>>>> high risk of overfitting (given high order polynomial)

J_o: is small?
J_train: is small; (as J_o and J_train fit training set well with high order polynomial)
J_cv: is high

actually without regularization, so hypothesis performance do not chagne : underfitting or just right or overfitting??-decided by polynomeal degree d

overfitting!! -decided by polynomical degree d.

> 2. λ is big: heavy penalizing parameter θ
>>>>high risk of underfitting (θ all close to 0, hypothesis nearly a flat straight line ~=θ_0-which not involved in regulization**n)**

J_o : with regulization term=big, as underfitting.
(object is to minimize J_o: while first part is J_train-high, second part regulization term-small as θ close to 0

J_train is big : first part of J_o: underfitting, could not fitting training set well.
J_cv: is high.

4. Summary:

J_cv: always without regulizaiton term, is high in both underfitting or overfitting;
J_train: always without regulizaiton term, is high in underfitting, and low in overfitting;
Jo: consider regulizaiton term if have,should be low both underfitting or overfitting?

Jo is only used to selecting paramter θ/ train algorithm on training set.-->do not compare vs J_cv or J_train, which used for selecting regulization parameter or polynomial degree.

for some data set, could look at the **plot of the whole cross validation error**-, then either manually , or automatically try to select a point , that minimize the cross validation eror, and select the value of λ that corresponding to low cross valiaiton error.

when selecting regulization parameter λ, ploting a figure like right, helps understanding better what's going on and helps verfity that whether picked a good value λ or not.

10_83. Learning curves / variance_Advice for applying Machine learning

Ploting to check if algorithm is working correctly or to improve algorithm pefromance.
Very offten used to diagnostic if algorithm is suffering from bias, variance or a bit both.

Which condtion could have both bias and variance error?

note:
1. debug:
>>1.1 linear regression/logistic classification: plot cost vs iteration: decrease or not-->reflect gradient descent computation right or not
>>1.2 : neural network: check gradient numerically vs computed derivative by algorithm: as cost could not reflect when algorithm compute derivative wrong-may still decrease with iterations: cost function of parameter is not convex

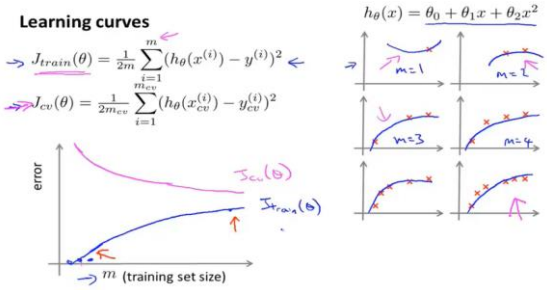
2. diagonostic: bias , variance
J_train, J_dev: work both for neural network and linear regression/logistic classification

1. Learning curves_m number of training example

1.1 Plot:
1. J_train: Training error: average squared error for linear regression on training set. or logistic cost function(or use average error times ?) without regulization term.
2. J_cv: on validation set. average squared error for linear regression on validation set. or logistic cost function without regulization term.

vs: m-number of training example.

(note:
1. for choosing hyperparameter based on test error on dev set:
test error : could use cost , acurracy, recall, precision, F1 , etc: -->to judge algorithm performance we cared, better or worse
2. for digonostic : bias, variance
could also use same test error : accuracy, recall, ...
)



train/ valiation / test error: J_train, J_cv, J_test
cost function without regularizaiton term on different set: training set, valiation set and test set. Note: for these three error calculation, Without regularizaiton term;

Cost function for training algorithm: J (θ. λ): consider regulization term.

1.2. Ploting process: m number of training example

for a selected hypothesis function (e.g : quadratic function like right pic):
like polynomial degree is already selected;

for each selected m : (take m to different value 1, 2, 3..etc.)
> train h(x) on m numberof training set by mininizing cost function J (θ, λ) , get trained paramter θ_train_m;

>compute J (θ_train_m) on validation set-->J_cv
and J (θ_train) on training set-->J_train.
--->training error and valiation error with only m number of training examples.
dev set should not be m size .

1.3. Ploting result: m number of training example

>1. if m is small---->over fitting
>>> J_train: small:
hypothesis function could fit training set perfect well when m is very small (eg.: m=1, 2, 3), with regulization term or not. J_train (θ_train) is small (every easy to fit single training example when m is small)

>>> J_cv is big:
trained hypothesis funcion could not generalize well to data did not seen before.

>2. if m is large ---->not like underfitting, as J_cv is small

H= θ^T * (x_1, x_2, ...x_m):
example x1, ..xm used to contrain parameter θ. **after certain size , new added training example could not add more contrain to parameter** : do no more contribute to example space matrix (x_1, ...x_m) : note m>n , matirx is singualar.-->adding more training example ~~do not help get parameter fit training set /dev set better~~ **could not add more constrain to algorithm parameters**.

>>>1. J_train: larger
when m become larger, it hard for hypothesis function h(x) to process through / fit all examples perfectly, J_train become larger.
--->actually J_train increase as m increase

>>>2. J_cv is become smaller:
only when get large training set, could the trained hypothesis function maybe fit data do not seen before better.
the more data you have, the better you do at generalizing to new examples.
--->actually _cv decrease as m increase.
(more training set-->training set features space could cover dev& test set feature-->algorithm fitting dev/test well)

>>> 3. J_cv, J_train will flatten out with m increase: J_cv _f ~= J_train_f

now training example space matrix (x_1, ...x_m) almost covers all dev/test set-sub example space, -->algorithm performance on these two example space almost same.-no matter undertaking, just right or overfitting-both large or small , or medium value.

note: even huge training size m , does not change the algorithm (polynomeal degree) property of underfiitng or overfitting for all training examples feature space.
with huge m, j_cv, j_train flatten out, and close to each other:
-->1. algorithm is undertaking: d is small-->J_train ~=J_cv, end up with big error value
-->2. algorithm is just right: d is just right-->J_train ~=J_cv, end up with small error value
-->3. algorithm is just overfit: d is very big-->overfit training set, dev set-->J_train ~=J_cv, end up with smaller error value.

Summary:

with m increase: J_train increase , J_cv decrease. as:
more training data, better hypothesis do at generalizing to new examples--> J_cv decrease
more training data , worse hypothesis do at fitting each signle training example-->J_train increase.

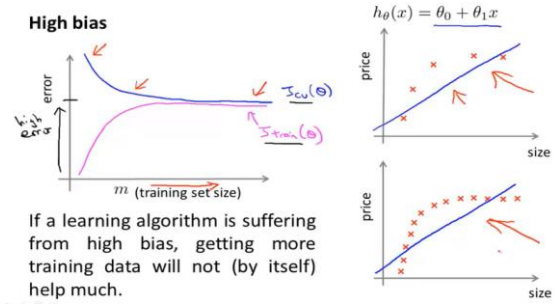
4. High bias (underfitting: example feature > hypothesis feature/degree d)

given hypothesis is linear funcion as right:

Increase training example number m:
will not change much hypothesis function -straight line (May same line gotten with less training example).
Just get the best staright line to fit training set, which could not fit this training data well.

--->J_cv do not decrease much: (when m=1, J_cv is large)
and when m come to certain value (could not add more feature/constrian to h(x)), you have almost fit the best possible straight line , which nearly do not change even m further go to very lager, -->J_cv plateau out/flatten out. (h(x) nearly no change, then it's error on test set no change)

>J_train: increase with m increase
J_train be small when m is small nearly 0, and then increase with m increase,--->-->in high bias case, J_train will end up to close to the J_cv_f - J_cv ~=J_train both high error



Note: in high bias case:
1.
as h(x) parameter number/feature/degree d is very small, when m is large, the preformace on training set and validation set will be very similar (when m is large , training set features have big possiblity including all test set features.-->h(x) error on training features, will similar on test features)

2. In high bias case, the final flatten value: J_cv_f, J_train_f are high values .
underfitting, h(x) polynomial degree/computed new feature could not learn good functions to fit data set, could not fix by increasing data size.

3. Increasing training example, will not help much high bias issue.
increased training examples, will end up almost same h(x) parameters which get in less training example.

---

5. High variance

high variance: hypothesis have high polynomial degree d-->many paramters/features > huge data features.

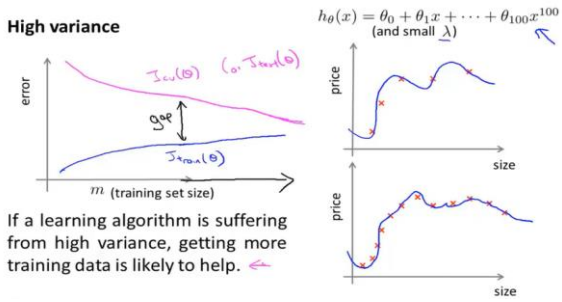
h(x) function like right pic: with very small regularization term-nearly no this term

>1. m is very small: overfitting.
J_train ~=0: h(x) fit training data very well with a function that overfit this
J_cv: very high as is most overfitting.

>2. m increase a little bit: -still -overfitting
J_train increase:
still overfitting training data, but less overfitting- slightly harder to fit training data perfectly.-->J_train increase.
but J_strain is small due to overfitting.

J_cv decrease much: as h(x) less overfitting with m increase, generalize better to set error with m bigger.
but J_cv is big value due to overfitting. and now gap between J_train and J_cv is big.
(trained h(x) only fitted to few data set feature included in training set, could not better generalize to test feature-->error gap between training error and test error is big)

>3. m increase - size including all example features (still less than h(x) degree):-overfitting
J_train increase:
still overfitting training data, but less overfitting- harder to fit training data perfectly.-->J_train increase.
J_cv decrease much: as h(x) less overfitting with m increase, generalize better to dev set with m bigger.
(note: training set feature space: is cover more and more of dev set and test set)



>4. m is large:-still overfit
increased training set could not add more constrains/features to hypothesis paramter, h(x) paramter θ almost no change
-->J_strain come to flatten value J_strain_f (not big as still overfit)
-->J_cv come to flatten value J_cv_f, close to J_strain_f
h(x) could fit almost all data set features, overfitting, -->should also overfit test set-->J_cv_f is also small and close to J_train_f.

with m increase , J_train increase slow (J_train_f small as still overfit), J_cv decrease much (close to J_train_f)

Summary :

in high variance
1. Getting more training data is likely to help.
esp. training data that could have new features/constrains to h(x) parameter θ/degree d.

high variace: h(x) paramter/feature/degree is very high, while **training data volume is small/do not have engough constrain to each feature's weight. (data set feature number>> data volume-data that can add constrain to feature weight)**
getting more training data-->mainly increase traning data feature space: nxm: -->extrem case: training set (x1, ...xn):线性无关， ->nxn space , 基向量-->cover any futher new example feature: x(k)=[x(k)_1, ..x(k)_n]
when training data including all data set feature, adding training dat make no sense. training data space matrix (x_1, ...x_m) is singular after certain size / value of m.

Note:
real ploting curve of J_cv, J_train may be a little noise, messier than above ideal case. But ploting like this could tell if algorithm have bais or variance or both problem.

10_84. Decide what to try next_Advice for applying Machine learning

1. Debugging a learning algorithm

Problem: hypothesis on new set , have large error in its predictions (J_cv/J_test is big).-->improve algorithm

Method:
>Get more training examples --->only fix high varaince
plot error pic and figure out have at least **a bit of variance:** means J_cv is quite a bit bigger than J_train

>Try **smaller set of features** in hypothesis--->only fix high varaince
plot error pic and if find have a high variance, carfully select out small set of features to use.fewer features could help.

>Try getting additioanl features in hypothesis:--->fix high bias (underfitting)
current hypothesis is too simple (feature/degree to small), get additional feature in hypothers to make it fit better to training set.
plot error pic and if find have a high bias
e.g for predting house price: get more info. about the pieces of land .and so on.

>>before doing above things, helpful to know in advance if this is going to help.

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

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

>Try adding polynomial fatures (x1^2, x2^2, x1*x2, etc): fix high bias
another way of adding features in hypothesis.

>Try decreasing regulation parameter λ-->fix high bias
decrease penalty on h(x) paramter, so to fit better on training set and test set.

>Try increasing regulation parameter λ-->fix high variance

Note:
1. High variance problem:
>Definiation: big gap between J_cv and J_train. Hypotheis funciton performce on training set and test set has big difference;

>Root cause:
~~Training set feature number is not big enough, could not cover test features;~~
Too many data features considered in hyposesis (high degree), yet training set has no enough constrasin to feature weights.
or algorithm captured additional fake features of training set , which not appliable on test set feature
(some features' weight on fit training set, which has no enough constrain..)

>solution:
>1. Increase training set (new feature vector):-->esp.与现有training set feature vector 线性无关-->increase feature vector space-->adding more constrain to feature weights
>2. if increasing traing set could not improve any more, then try regulization -reduce feature weights (esp. for the 'fake features')
(training set big enough--> feature vector-->featur matrix nxn, if still overfitting, means there are feature x_k 与其他feature 线性相关， feature space < nxn, feature matirc no enough constrain for feature weights
--->reduce fake features' weight

2. High bias problem:
>Defination: big J_train. Hypothesis could not even fit training set well.

>Root cause:
Trained hypothesis model could not capture enough training set feature.
(data set feature number used in hyposesis is too small -->polynomial degree small, no enough to capure data interesitng property)

>Solution:
>1. mode architecture: try big polynomial degree, more layer, more units in layer;
>2. iterates more steps, get better h(x) paramter θ.

---

2. Neural networks and overfitting

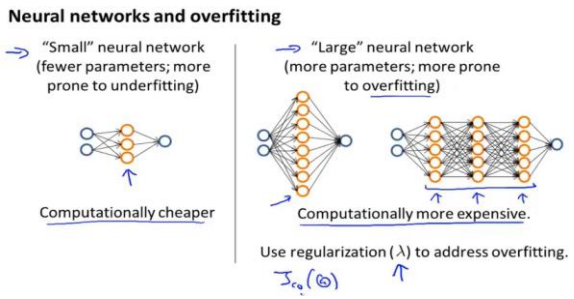
Neural network pattern choosen.

2.1: 'Small' Neural network: (few new created features) prone to underfitting
small network: few layers, and fewer units in each layer.-->few parameters, more prone to underfitting

computionally cheaper

2.2: **'Large ' Neural network, (more features, prone to overfitting)**
Large network: more hidden layers or more hidden units in layers.

computationlly more expensive.
Normally the larger network, the preformance better. could use regulizaiton to address overfitting.



Note:
> **Normally, Large neural network + regulizaiton more efficent than small neural network.**

>>> Hidden layer number selecting:
same as selecting polynomial degree/ regulizaiton paramter, try neural network with different layers. for each choosen layer neural network: training set, validation set, test set.
train algorithm on training set-->calculate J_cv error on validation set --->selecting the lowest J_cv, and corresponding layer ---->report J_test for the choosen layer neural net work error on test set.

Summary_linear regression/logistic classification

for a given size training example m:

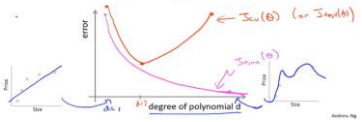
1. polynomial degree d influence:
algorithm performane will change from underfitting to overfitting , when d increase from small to large

if overfitting-->could reduce d: dropput,

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 $J_{test}(\theta)$)



for a given size training m, and selected polynomial degree d_s.

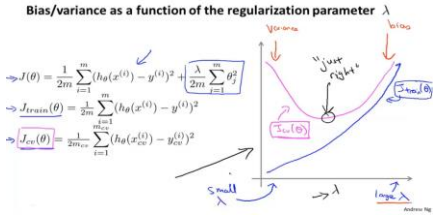
2. Regulation parameter λ infuence:

algorithm performance will change from 'overfitting' to underfilling with parameter increase (reducing paramter θ from huge large to 0, there is just right λ to make just right j_cv.).

'overfitting' :regulation parameter=0, no regulization term.
underfitting: regulation paramter high;

but if d0 degree polynomial hypothesis is underfitting itself, then peformance change should start from from 'underfitting' to worse underfiting with λ incease.

Bias/variance as a function of the regularization parameter λ



if overfitting (high polynomial degree d)-->increase regulation term (right λ) could help high virance /reduce J_cv to just right point.
if underfitting-->decrease regulation term (right λ) could help reduce high bias probem

if without regulation, h (x) with selected d_s, m training set,
>1. Underfitting : h(x) feature less than trainng set
>>add regulation term λ_0, will penalize h(x) parameter, --> trained h(x) now worse fitting training set and test set-->add regulation term to undertaking make J_train increae and J_cv increase (still underfitting)
>> decrease λ_0, reduce penalizing on paramter, h(x) better fitting on training set and test set. (still underfitting)
J_train decrease and J_cv decrease (still underfitting)
>>increase λ_0, J_train increase and J_cv increase (still underfitting)

>2. overfitting: h(x) feature more than training set feature:
>>add/increae regulation term λ_0, will penalize h(x) parameter, --> trained h(x) feature weight smaller, worse fit each training set all feature and but better fit on test set feature
--> J_train increase and J_cv decrease (if parameter is not 0, then end up still with a little bit overfit, J_train_f is very small, while J_cv_f close to J_train_f)
>>decrease λ_0, more overfitting J_train decrease and J_cv increase (still overfittiong,J_train not much)

for selected polynomial degree d_s.

d_o= just right polynomial degree for huge traning example

3. influence of training example number m

J_strain: increase from very small to large till nearly flatten , with m increase;
J_cv: decrease from high value continuely then to faltten , with m increase

if d_s > d_o:

J_strain: increase from very small to large till nearly flatten , with m increase;
J_cv: decrease from high value continuely then to faltten , with m increase

>m: 1-just right number (constran h(x) feature d):
J_strain increase from nearly 0, J_cv decreae from high value

>m: continue increase: could not add more feature, both J_stain and J_cv come to flatten value: these two value close, but normally J_cv_f slightly higher than J_train_f.

if d_s > d_o:

note:

when overfitting: increase training example could help.
as when overfit, example feature number involved in hyposesis is too many, not enough training set volume not big enough /feature matrix (x1, ..xm) no big engough to give constrain for every feature/hypothesis paramter.
Add training example number, especailly example that add new features vector (add new constrain -与现有training set/feature vector 线性无关), could help overfittiong;

when underfitting: increasing training example could not help.
as underfit: example feature number hypotheis degree/feature too small, adding more training example may slightly increase J_train, decrease J_cv slightly , not much help

if d_s<d_o:
J_strain: increase from very small to large till nearly flatten , with m increase;

>>m: 1-just right number (constran h(x) feature d):
when m is very small, over fitting, training example is fitted perfectly J_starin almost 0. when m number increase, less overfitted-fitt training example less perfectly-->J_strain increase.
when m increase to a certain value, that could just constarin d_s (polynimial features) , get the trained h(x)_θ_o'

>>m: just right number--->large number: just have all example feature
m featue > h(x) feature, kind of underfitting. but compare to above h(x), current h(x) is learning to fit more example features, and fit less well on all training example than it did on previous smaller training example which have less features , but could fit better on test set features.-->, J_train continue increase

>>m: continue increase:
at this time h(x) paramter θ nearly no change as new added example do not add more constrain: .-->J_strain come to nearly flatten value.

J_cv: decrease from high value continuely then to faltten , with m increase

with m number increase, h(x) have been trained to fit 'huge training example' better-->error on test set is becoming less.

>> m: 1-just right number (constran h(x) feature d):
J_cv decreae singly, as h(x) is fitting better on test set;

>>m: just right number--->large number: just have all example feature
m featue > h(x) feature, kind of underfitting. but compare to above h(x), current h(x) is learning to fit more example features, and fit less well on all training example than it did on previous smaller training example which have less features , but could fit better on test set features (test set could be any some features of data).-->, J_train continue increase, and J_cv continue decrease

Note: from nowon, h(x) with d_s degree is actually underfitting: could not solve underfitting problem with m increase:
increasing m will only add more example features, while h(x) learned to fit all features while without enough ability (h(x) degree not high engough)--->
J_cv will decrease but have bottom line (flatten value)-do not decrease much, and J_train will increase (also have top line)when increasing m.

>m: continue increase:
at this time h(x) paramter θ nearly no change as new added example do not add more constrain. h(x) error on test set nearly no change either- ->J-cv come to a flatten value : normally this value J_cv_f close to J_strain_f, but slightly higher.

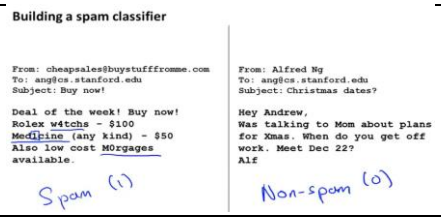
W12 - Machine learning system design

12_85. Prioritizing what to work on: Spam classification example__Machine learning system design

Foucs on Issues faced on Machine learing system design. **Prioriting your time for what to work on.**

1. Building a spam classifer

using supervised learnign to distinguish between spam and non-spam based on training set (labeled as spam or not)

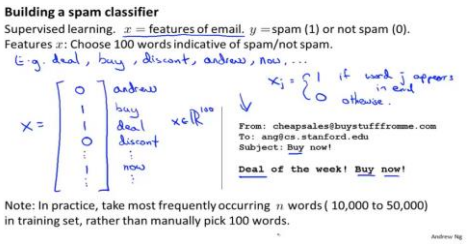


2. Feature represent_for training set

>first: represent x as features of the email (while y=1 (spam) or 0 (not spam)):
represent method:
e.g: choose 100 words indicative of spam /not spam.
When there are those words in email-->judged as spam/not spam.
E.g when there are words'now' in mail, judge as not spam.

> >create feature vocabulary/dictionary : dimenion is the feature words number/size

>> create feature vector X for each mail: dimension is same as feature dictionay ;
0 / 1 in feature vector means corresponding, same positon feature word in feature dictionay , not occure /occure in email.

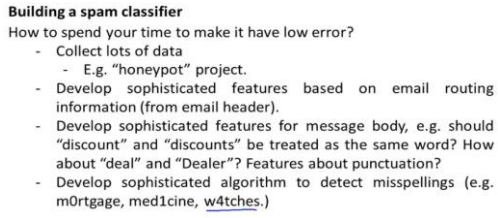


Note: for feature words choosen:
normally look at the training set and take most frequently occuring words (10,000 -50,000) in training set, rather than manually pick 100 words.

3. Priority for making classifier having lower error

>1. Collect lots of data
e.g: 'honeypot' project:
creat fake emal address and get them into the spammers hands, then collect tons of spam email.

>2. Develop sophisticated features based on emali routing info. (from email header).
spammer often will try to obscure email origins using maybe fake email addres or send through very unsual sets of compute service..etc. some of these info. will reflect in email header.-->new feature based on email header to caputer email routing info. is an efficent way to identify if it is spam or not.



>3. Develop sophisticatd features for message body.
e.g how about 'deal' and 'dealer', treased as same word? spam may have particular favor for word/punctuations.

>4. Develop sophisticatd algorithm to detect/correct misspellings
e.g: 'w4tches' in mail, spam classfier might not equate this as the same thing as the word 'watches' (which choosen as feature word), harder to realize this mail is spam with these deliberate imsspellings. that's why spammers do it.

use error analysis , to tell where can try to **have a more systematic way to choose among options of the many different things you might work on.**-->More likely to select what is a good way to go for further step.

12_86. Error analysis__Machine learning system design

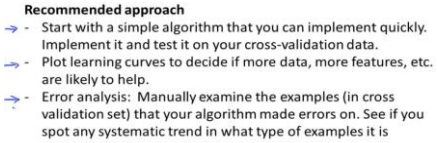
1. Recommended approach

>1. Start with a simple algorithm that can implement quickly.
Implement it and test it on cross-valiatio data.

Usually spend 1 day to try get something quick and dirty , and implement it and test on dev set.

>2. Plot learning curve to decide if more data, mroe features, etc. are likely to help.
figure out if have high bias/ variance or both using plotted curve.-->then try corresponding method to improve.

when just start machine learning problem, it is very hard to tell in advance whether need more data or more feature in the beginning without learning curve.
by running even a very quick and dirty implementation, and by plotting learning curves, that helps make decisons where to go.--->let evidence rather than feeling, guide on where to spend time

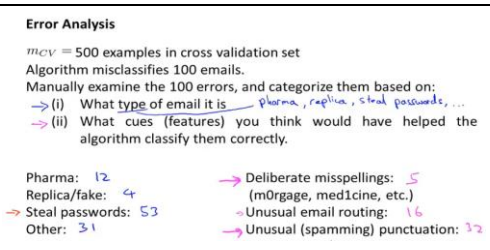


>3. Error analysis:
manaully examine the examples (in cross validation set) that algorithm made error on.
see if you spot any systematic trend in what type of examples it is making errors on.
-->inspier to design new features, or tell whether current things /shortcomings of the system etc. give inspiration you need to come up with improvements to it.

2. Error analysis

e.g: m_cv=500, algorithm make **error on 100 validation example.**
-->
Manually examine the 100 **errors, and categorize them based on:**
>Email type
counting up emails in different catagory, -->work on catagory that have more errors.
e.g; sell things (20pc), steal password(70pc)..-->work on steal passoword detetcing.

> Cues/ featuers of these mail, that may have helped the classifier class them correctly.
counting up emails in different features, -->work on features that have more errors.
e.g: deliberate misspellings (5pc), unusual email routing (16 pc), unusual punctuation (32 pc)-->work on email routing and punctuation.



3. Principle

Implement simple algorithm quick and dirty-->find the most difficult examples to identify and focus effert on those.

Note: Different algorithm almost always find similar categories of examples difficult.

4. Importance of numerical evaluation
Incredebly helpful to have a way of evaluation learning algorithm that gives back a single rule number (maybe accuracy , maybe error) tell how well learning algorithm is doing.

Summary:
when developing new algorithm, need to try lots of new ideas and lots of new versions of algorithm. if every time try new idea, if end up manually examing a bunch of examples again to see better or worse, makes harder to make decison for this idea effect on algorithm performance.

single rule number could let you just look and see if the error go up or down, use it more rapidly to try out new ideas, and almost **right away tell if new idea improved algorithm or not.**

Note: do error analysis on dev set, instead of test set.

(note:
1. train/dev/test error： algorithm performance on training/dev/test set
2. performance/error representation could use: cost , accuracy, recall, precision, F1 , etc.
3. error usage:
>>: check bias/varaince
>>: as signal evaluation to choose new ideas/features
>> choose hyperperameters based on cost on dev

4. cost funtion vs iteration : usage:
-->debug
(only for linear regression, logistic regression, neural network use gradient check for debug as cost vs parameter is not convex)
--> if converged or not

The importance of numerical evaluation

Should discount/discounts/discounted/discounting be treated as the same word?
Can use "stemming" software (E.g. "Porter stemmer")
universe/university.
Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.
Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.
Without stemming: 5% error With stemming: 3% error
Distinguish upper vs. lower case (Mom/mom): 3.2%

E.g:
evaluate using 'stemming' software to treate ' discount' and 'discounted' as same words:
Error analysis may not be helpful for decidiing if this is likely to improve preformance. Only solution is to try it and see if works.

Singual rule number is dev error:
run algorithm with stemming and without stemming sperately on dev set, -->compare dev error J_cv. then decide to use stemming or not.

12_87. Error metrics for skewed classes__Machine learning system design

Error metrics: a single real number evaluation metric for learning algorithm to tell how well it's doing.

Skewed classes: particularly tricky to come up with an appropriate error metric.

1. Skewed class example:

e.g: trained classfier have 1% test error (tested on test set)
while only 0.5% of test set actually have canceer.-->algorithm error 1% is not good.
If take unleraning algorithm: just let y=0 ,given whatever input x, -->test error then is only 0.5%, better than above learning algorithm

(note: for skewed class--->use unsupervised learning: anomaly detection, positive y=1 is small volume, save for dev and test set; training set is y=0)

Cancer classification example
Train logistic regression model $h_\theta(x)$. ($y=1$ if cancer, $y=0$ otherwise)
Find that you got 1% error on test set.
(99% correct diagnoses)

Only 0.50% of patients have cancer.

skewed classes.

```
function y = predictCancer(x)
  y = 0; %ignore x!
return
```

0.5% error
99.2% accuracy (0.8% error)
99.5% accuracy (0.5% error)

2. Using accuracy/error evaluating skewed classifier

Skewed classes: ration of positve to negative example (in data: including training, dev, test) is very close to one of two extrems (postive is much more or negative is much more than the other class)

in this case unlearning algorithm (let y always =1/0) could do pretty well on prediction.

using classfication accuracy / classification error (cost on dev) as evaluation metric for skewed class:
Even **result in high accuracy /low errors, do not always clear if doing so is really improving classifier quality.** like righ e.g

e.g:
accuracy improve from 99.2% (0.8% error) to 99.5% (0.5% error):
may just simplify algorithm just let y=0 or 1 (accuracy higher, yet algorithm itself not improve /particularyly good calassfier)

3. Precsion/ Recall_for skewed class

Defination:
True positive: test data class is positive (1), and algorithm prediciton is also positive (1);
True negative: test data class is negative (0), and algorithm prediciton is also negative (0);
False Positive: test data class is negative (0), yet algorithm prediciton is positive (1);
False negative: test data class is positve (1), yet algorithm prediciton is negative (0);

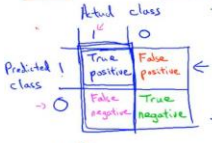
Precison: true positive / predicted positve(true positive + false positve)
(of all test set that predict positive, what fraction actually is positive)

Recall: True positive /actual positive (True positive + false negative)
(of all test set that is actually positve, what fraction did algorithm correctly detect as positive)

for unlearing algorithm y always =0 (above e.g),- true positive=0->presion and recall=0

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?)

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

Note:
1. when defining precsion and recall, usually use the convention that **y=1 in the presence of the more rare class.**
2. Precison/Recall metrix is a good way to tell if algorithm even for skewed class is doing well or not.
3. for skewed classed, precison/recall is a better way than accuracy /error for evaluation algorithm performance.
can prevent unlearning algorithm 'cheating' like y always =0 which have high accuracy/low error.

for other un-skewed data, which evaluation value used often : accuracy / cost

11_88. Trading off precision and recall__Machine learning system design

1. Matchmatic intuiation of precision and recall

Increase precsion-->recall decrease
Intuition 1: increase precision: tend to make algorithm predict positive more strickly,
--> predict less positive, less true postive , and much less fasle positive (as positve defined by rare class, negative have much more examples), precision increase.
-->predict more negative, more false negative: less true positve-->smaller recall

Intuition2:
-->increase N_t/P_t, -> decrease p_t slightly , while N_t decrease bigger (as postive define is the rare class-small P)

Increase recall : -->Precsion decrease

Intuition 1: increase recall: tend to make algorithm predict positive less strickly,
--> predict more positive: more true postive , and much more fasle positive (as positve defined by rare class, negative have much more examples), precision decrease
-->predict more true positive,-->big recall

Intuition2:
--> P_t ↗, N_t ↗ (bigger than p_t increment as postive define is the rare class-small P)--> increase N_t/P_t-->precsio decrease

Root cause: big volume diffrence for positive and negative.
Positve is defined as rare class, small volume. when algorithm improved to increase precsion, conditons to judge as postive is more strict: both true positive number and false positive number decrease . but false positive number decrease much bigger than true positive.

Ture positive number decide recall, -->decrease.

test set number =P (actual positive) + N (actual negative) ,
P_t= true positive number;
P_f= false positive number;
N_t=true negative number;
N_f= false negative number;

P, N do not change, while p_t, p_f, N_t, N_f changes when algorithm change.
P+N =P_t+P_f + N_t+N_f;
P=P_t + N_f
N=N_t + P_f

Precsion = P_t / (P_t + P_f)= 1/ (1+ P_f/P_t))
Recall = P_t / P

P_t ↗, P_f ↗↗↗, -------->>precision decrease, recall increases
P_t ↘, P_f ↘↘↘↘, --------->>precision increase, recall decrease

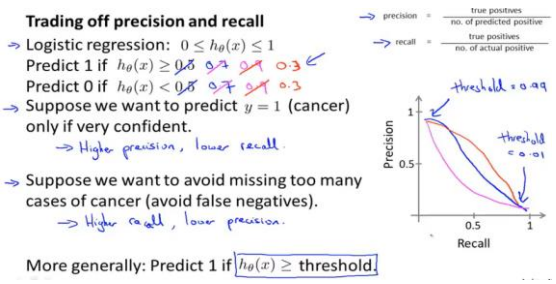
2. Trading off precsin and recall_Example

2.1 cancer detection: increase precsion (for predited positves, the factor of actuall positive)
only predict y=1 when have high confidence:
predict y=1 , if h(x) >=0.5--->= 0.7 / 0.9
predict y=0 , if h(x) <0.5--->, < 0.7 / 0.9

with this high confident h(x)=0.9, precsion is higher.
Yet smaller number of patiens will predicted as y=1, **recall is smalle**r (won't correctly detect that those patiens is labeled as 1).

2.2 Cancer detection: increase recall (incase missing too many labeled 1 in test set, avoid false negative)
predict y=1with lower confidence:
predict y=1 , if h(x) >=0.5--->= 0.3
predict y=0 , if h(x) <0.5--->, < 0.3

with lower confidence h(x)=0.3, recall is higher.
as correctly flagging a higher fraction of all labeled 1 test sets: more patiens will be predited as y=1)
Yet precsion is smaller:
as high fraction of the patiens predicted as y=1 (true positve + false posive), then high fraction of them turn out actully labeled as 0. (false positive is high): (true positive / ture + false positive) is smaller.



Varisy threhold of confidence h(x) , to trade off precsion and recall.

Ploting precision and recall: have many different curve shape depending on the classifier detail.

3. F1 Score_

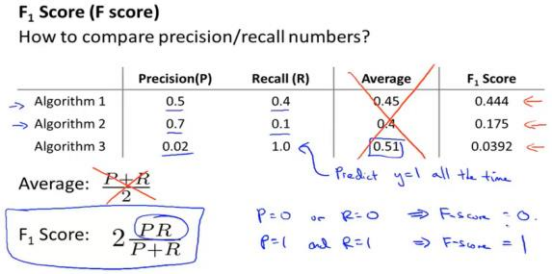
compare diffrent presion and recall for diffirent idea/algorithm

>1. Average: (P+R)/2--not a good way for evaluating algorithm

for threshold two extrems (P=1 or 0), both algorithm is not good , yet average (P+R) is not too bad.
in right pic. example, algorithm 3 is having recall =1 (y=1 unlearning algorithm), not good algorithm, yet average (P+R) /2 is higher than other two algorithm.

>2. F1 score = 2P*R/ (P+R)
A little like taking the average of precison and recall, but gives a higer weight to the lower value of precison and recall:
either P or R is 0, F1 will also be 0.
Want F1 be large, then both P and R need to be large.

---F1 gives reasonable rank ordering of different classifiers.
---have many different formula for F1, above is more often use.



4. Choose threshold automatically for balanceing precison and recall:

try a range of threshold value. try different threshold, for each threshold, do like selecting regulazation parametr/ polynomial degress / training example number:

> train algorithm (y= 1 if h(x) >= threshold) on training set-->alogorithm parameter θ trained;
have this step or not?-->no
does different threshold infunce paramater θ? or just try to choose threshold value for alreay trained/ pretrained algorigthm.
for trained algorithm (with optimizing object: threshoh not used optimizing parameter), use different threshohd on dev set -->pick the threshhold that achieve the highest F1 score.

>test trained algorithm on dev set;
> compute F1 score for this algorithm on dev set.

>compare F1 for algorithm with different threshold value, chose the biggest one/threshold/algorithm

12_90. Data for machine learning __Machine learning system design

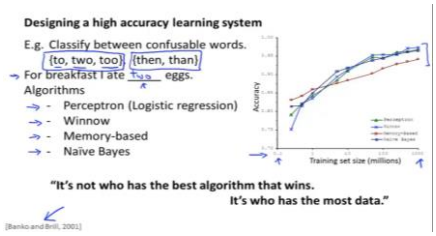
How much training example to work on.
Under certain conditions, getting a lot of data and training on a certain type of learning algorithm, can be a very effective way to get a learning algorithm to do very good performance.

1. Example: diffrent algorithm training on difference training set

vary training set size, and trained different algorithms on the range of training set sizes
example conclusion:
1 Most of these algorithms give remarkely similar performance.

2. As training set size increase, from few hundreds to thousand millions, the preformance of algorithm all pretty much monotonically increase.
even the 'inferious algorithm', if give it more data, then most likely it will beat even a 'superior algorithm'

give a trend : "it's not who has the best algorithm that wins. it's who has the most data.'



2. Large data rationale

Conditons massive training set that will be able to help.

2.1 Feature x have suffcent info. for specific y-prediction.

~~training set feature could cover all test set feature;~~
~~does it also mean: even small traing set cover all the data set features?~~
~~even 1/2 training sample have show all the data features, need more/enough training data to constrain hypothesis paramcter.~~
~~training data need to :~~
~~1. have engough feature cover all data set feature;~~
~~2. have enough constrains to hypothesis paramter/degree/feature;~~

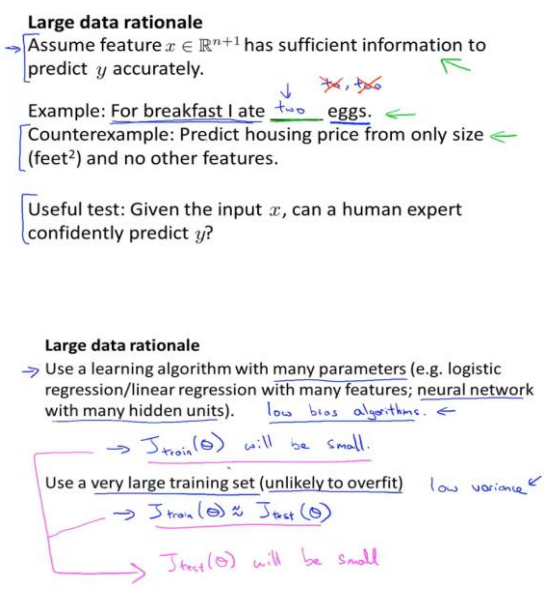
-->
1. for linear regression/logistic classification: y/decision boundary = Φ_0+Φ_0*x_1 + Φ_2 *x_2 + ...+ Φ_n*x_n
   have engough feature info. x_1, ..x_n (for each example), to caputure training extrample info. to predict y
   feature/polynomial degree is just fit or overfit training & dev & test set
2. for neural network: enough final 'new created feature' feeding to last layer-->
   input feature vector should be engough, and neural network architecture: hidden layer number, hidden uits should be big enough.

Hypothesis function need:
1. have engough parameter to capute all training set feature;
2. Capture less fake feature from training set, while not applicable to test/set.

And use a **learning algorithm with many parameters (high feature number)-low bias algorithm**
e.g: logistic regression / linear regression with many features; neural network with many hidden units;

> J_train will be small (overfitting)
> using very larege training set , will be unlikely to overfit (when training example number large ~~than hypothesis parameter number, feature space bigger,~~ better cover dev/test set feature vector)
--->and J_train_f~=J_cv_f / J_test_f is small

Adding more training set~~more features in training set~~, add more constrain to hypothesis, make less overfit, to just right point .
J_train increase to flatten, but still small as a little bit ovefit in last;
J_cv decrease to flatten, close to J_train (as training set cover all test set feature. only hypothesis may have more 'fake features' captured in training set , that not applicable for test set-->J_cv_f a bit higher than J_train_f



Algorithm that no high bias, and no high variance:

No high bias: using algorithm with many paramters (high degree/many features)
No high variance: using huge large training set: (algorithm could genaralize well from trainig set to dev set,-->traning set is large enough to cover all dev/test set features)

1. Key point for geting low bias on test set:-->a little bit overfit

>1. hypothesis funciton high degree/data set features ;
>2. training set big engough -->have enough features that cover all test set feature;

2. Get low variance on test set: huge training set.-->a little bit overfit
(training set have all features could cover test set , and these features captured by hypothesis function. while also try to make the 'fake features' caputured by hypothesis, which could not apply to test set, as small as possible)

huge data to train algorithm with many parameters might help get a high performance learning algorithm.

Summary:
Pre-judage,
> **first , if human expert could predict confidenly based on these training set/feature**.
This is a sort of certifiaon if y can be predicted accurately from input feature: input feature is enough, algorithm also have enough parameters.

> 2nd: if can **get large training set, to train algorithm with many parameters.** : to make training example feature space (x1, ...xm), nearly cover all test example.

if can do both, then could get a good performance algorithm