Introduction to Machine learning - Andrew Ng.

Lec 1.1

T: Task → Classifying emails as spam.
P: Performance measure. → Correctly classified emails.
E: Experience. → watching you label as spam or not.

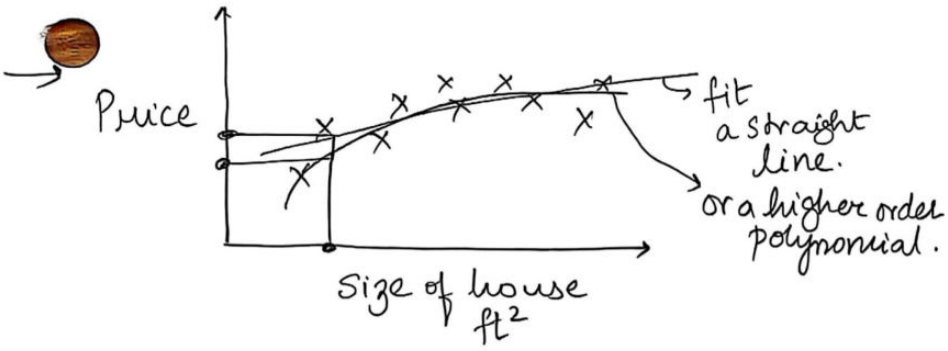
Learning algorithms.

① Supervised learning & ② Unsupervised learning.

Others: Reinforcement learning.

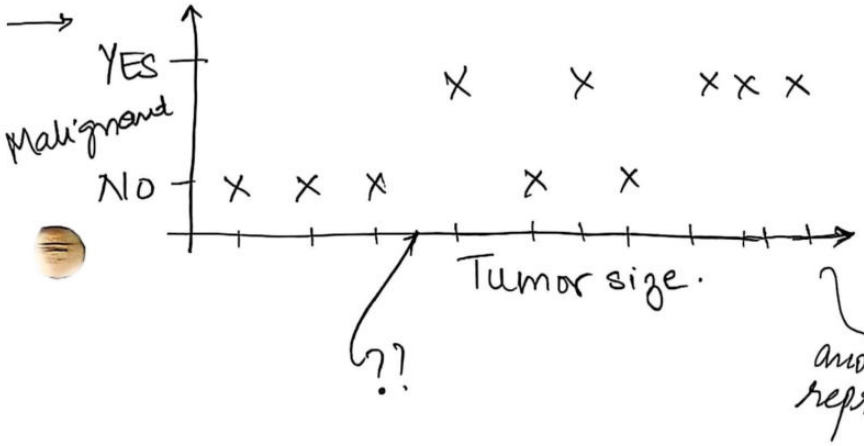
Lec 1.2

Supervised learning.



→ "Right" answers given & we want a function at all values.

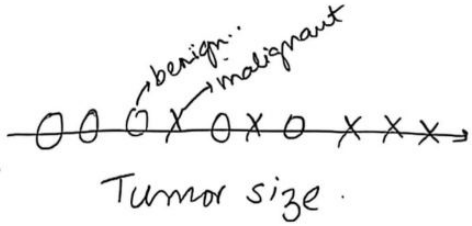
aka **Regression problem** :- Produce a continuous valued o/p.

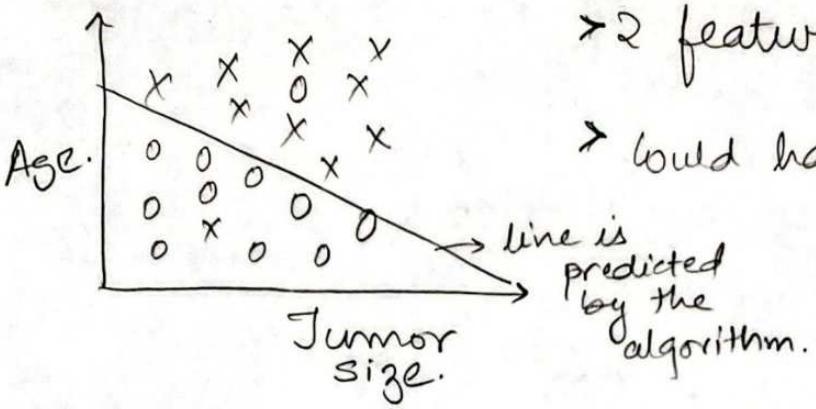


Classification problem.

↳ Could have multiple classes.

another representation.





> 2 features { > Age.
> Size of tumor.

> Would have multiple features.

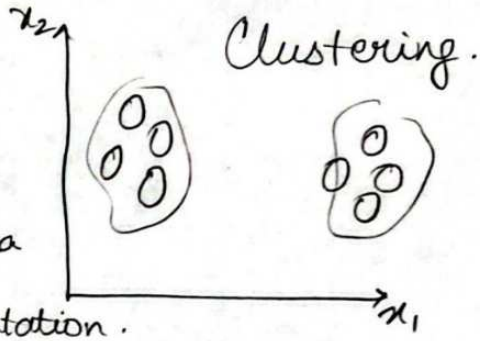
> Support vector machine (SVM) algorithm's can deal with a no. of features.

Lec 1.3

Unsupervised learning.

> No labels!
> Find structure in data.

> Clustering algorithms → Google news!
→ Astronomical data analysis
→ Market Segmentation.



> Cocktail party problem.

Speaker #1 ⟹ Microphone #1
Speaker #2 ⟹ Microphone #2

} Cocktail party algorithm. can seperate these two sources.

> One line of code: wow! → Uses SVD!
> Use Octave or Matlab.

Lec 2.1 Linear Regression

Supervised learning.

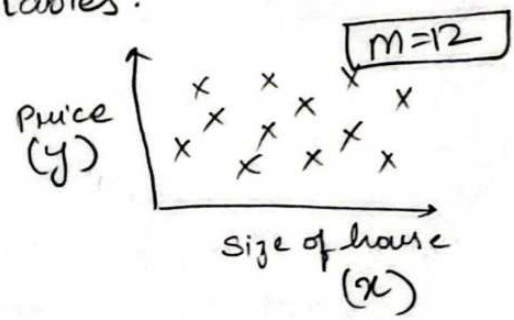
- Given a training set.

Notation

m = no. of training examples.
x = input variables / features
y = output variables / "target" variables.

(x, y) → one training example.

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

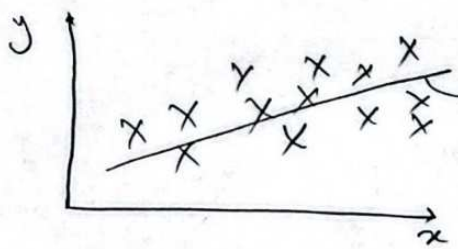


- Training set
- Learning algorithm
- new I/P → Hypothesis 'h' (function!) → Predicted value (estimate).

- h maps from x to y!
- How do we represent h?

$h_\theta(x) = \theta_0 + \theta_1 x$. ⇒ y is a linear function of x

↳ shorthand: $h(x)$.



↳ This model is called linear regression with one variable.

⇒ Univariate linear regression!

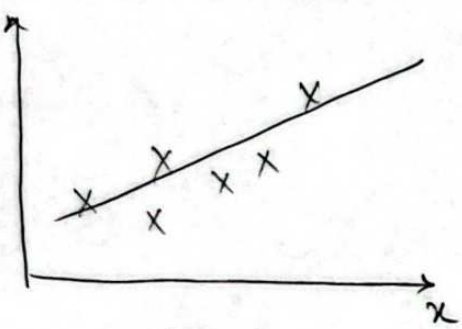
lec 2.2 Cost function

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

where θ_i's: are called parameters!

Choosing θ_0 & θ_1

θ_1 = slope & θ_0 = y intercept.



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

> Minimize $\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ ⇝ Objective function!

Minimize θ_0, θ_1

Find $\theta_0 \& \theta_1$ such that expression is minimized

Cost function → AKA Squared error cost function.

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

⇒ Minimize $J(\theta_0, \theta_1)$
$\theta_0\ \theta_1$

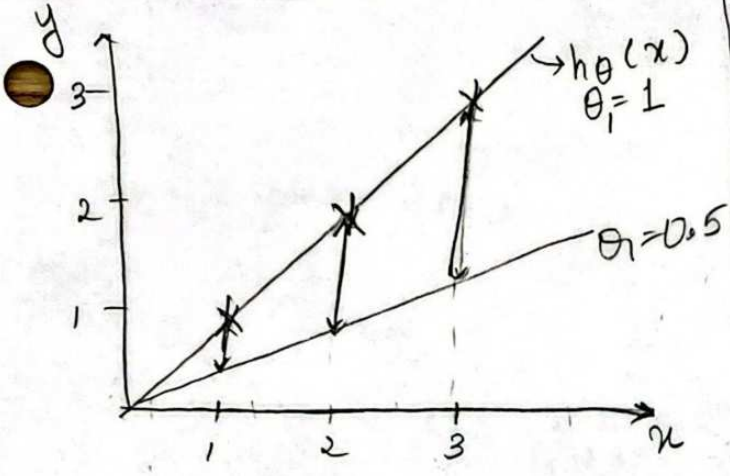
> Squared error cost fu is most commonly used & works well.

lec 2.3 Cost function intuition I

Eg:- Using a simplified $h_\theta(x) = \theta_1 x$

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

$h_\theta(x)$



$J(0.5)$ = sum of squares of vertical lines.

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

$$\cong 0.58$$

$$J(0) = \frac{1}{6}[1+4+9] = \frac{14}{6} \cong 2.3$$

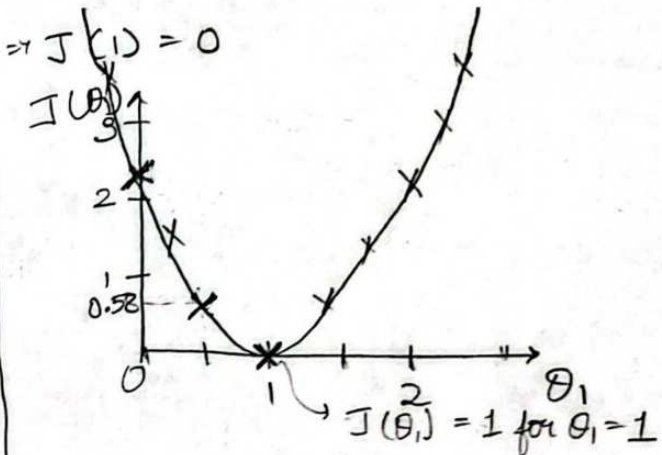
$J(\theta_1)$

$J(\theta_1)$ when $\theta_1 = 1$?

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

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

$\Rightarrow J(1) = 0$

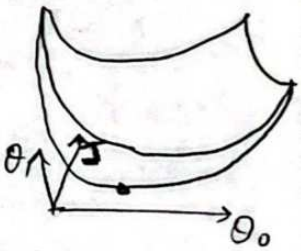


> Each value of θ_1 gives a different value of cost function, which we need to minimize.

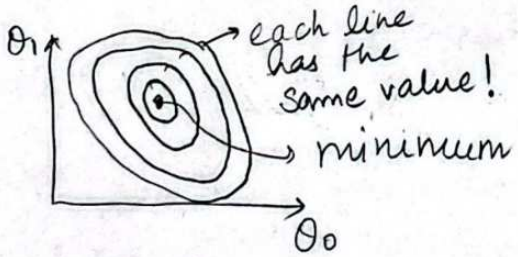
Lec 2.4 Cost function intuition II

> Given $h_\theta(x)$ gives $J(\theta_0, \theta_1)$. if $\theta_0 \neq 0$.

> With θ_0 & θ_1 the plot of $J(\theta_0, \theta_1)$ is 3D → Bowl!



> Use contour plots!



Lec 2.5 Gradient descent

> Minimizing an arbitrary function $J(\theta_0, \theta_1, \theta_2 \dots, \theta_n)$

> Outline

- Start of with some θ_0, θ_1 (say $\theta_0 = 0, \theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we reach a local minimum.

Gradient descent algorithm.

repeat until convergence {

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

}

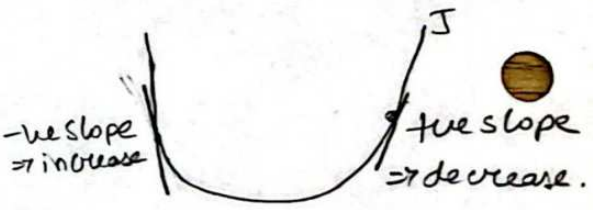
Correct: Simultaneous update.

$$\text{temp}0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp}1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$\theta_0 := \text{temp}0$

$\theta_1 := \text{temp}1$



> $a := b$ is assignment! (or ==) $a = b$ is truth assertion.

> α is called learning rate!

> α is too small ⇒ grad descent is slow.

> α is too large ⇒ could fail to converge or diverge.

> α could be fixed & it could still converge since $\frac{d}{d\theta_j}$ becomes smaller.

Lec 2.6: Gradient descent intuition.

> No notes.

Lec 2.7: Gradient descent for linear regression

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

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

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

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

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

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

=> **Algorithm:** → use simultaneous update!

repeat until convergence {

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

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

}

→ predicted value by chose $\theta_0 \& \theta_1$ for $x^{(i)}$

→ actual training output for $x^{(i)}$

> Cost function for linear regression is always a convex function ⇒ it has only one global optimum ⇒ grad. descent always converges.

> Algorithm used now is called 'Batch' gradient descent. Since in every step all the training examples are used.

Lec 2.8 Extensions.

> Could solve linear regressions directly called normal equation $A^TA\hat{x} = A^Tb$ which projects b onto col-space of A. But this does not scale as well.

> When we have larger input features, use linear algebra.

Lecs 3 Linear Algebra review!

> Not commutative $AB \neq BA$

> Associative $A(BC) = (AB)C$

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

Lec 4.1 Linear Regression with Multiple Variables.

features: x_1, x_2, x_3, x_4

output: y.

x_1	x_2	x_3	x_4	y
..				
..				
$x^{(3)} \rightarrow$				
..				
..				

n = number of features

m = number of training examples.

$x^{(i)}$ = input features of i^{th} training example. → it is a vector of dimension 'n'

$x_j^{(i)}$ = value of feature j in i^{th} training example.

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

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

In general: $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

Where $x_0^{(i)} = 1$

$\Rightarrow x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$ ($n+1$ dimension.) $\quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

$\Rightarrow \boxed{h_\theta(x) = \theta^T x}$ → feature vector.

θ^T → parameter vector. $\quad x$ → inner product. $\langle \theta, x \rangle$

Lec 4.2 Gradient descent with multivariate linear regression.

- Cost function

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

Gradient descent

Repeat {

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

} → simultaneously update for every j.

new algorithm.

Repeat {

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

}

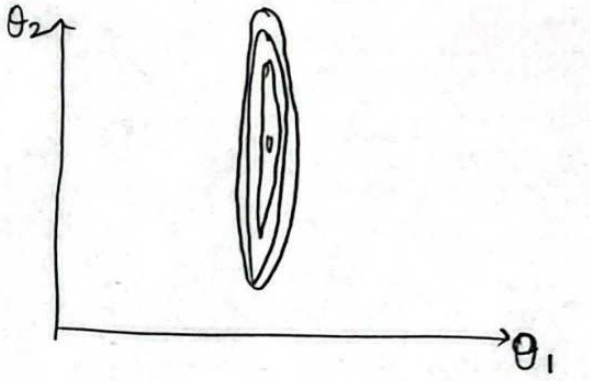
Lec 4.3 Feature Scaling

> Make sure features are on a similar scale.

x_1 = size (0 - 2000 ft^2)

x_2 = no. of bedrooms (1-5) ⇒

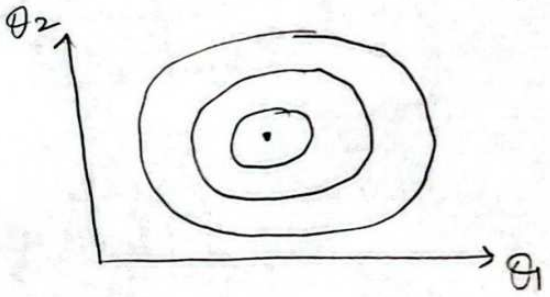
↳ Contours are narrow & gradddescent can take forever to converge.



→ **Normalize the features.**

$$x_1 = \frac{\text{size}}{2000}$$

$$x_2 = \frac{\text{no. of bedrooms}}{5}$$



Aka feature scaling

→ Try to scale features b/w $-1 \leq x_i \leq 1$ or close to this range. $(3, -\frac{1}{3})$ to $(3, \frac{1}{3})$ is a descend scale.

→ **Mean normalization.**

> Replace x_i with $x_i - \mu_i$ to make all features zero mean.

> Do not apply to $x_0 = 1$

Eg: $x_1 = \frac{\text{Size} - 1000}{2000}$ where size varies from 0 → 2000

$$x_i = \frac{x_i - \mu_i}{s_i}$$

μ_i → avg value of x_i

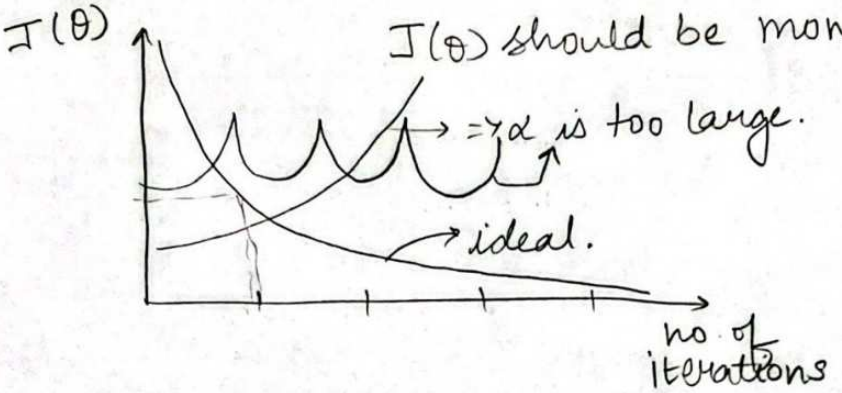
s_i → range (max - min) or standard dev.

Then, x_i roughly lies b/w -0.5 to 0.5

Lec 4.4 "Debugging"

> How to make sure gradient descent is working well.

> Plot $J(\theta)$ vs. no. of iterations.



$J(\theta)$ should be monotonically decreasing.

> α is too large.

> α is too small ⇒ slow convergence

> Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in an automatic convergence test. Plotting is better.

> Try $\alpha = 0.001 \overset{\times 3}{\text{---}} 0.003 \text{--} 0.01 \text{--} 0.03 \text{--} 0.1 \text{--}$

Lec 4.5 Polynomial regression ⇒ features.

Eg:

$h_\theta(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$.

Instead create a new feature $x = \text{frontage} \times \text{depth}$.

$\Rightarrow h_\theta(x) = \theta_0 + \theta_1 x$. ⇝ reduced the no. of features.

Polynomial regression.

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

We have $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x^3 + \ldots$

$\Rightarrow$ use $x_1 = x$

$x_2 = x^2$ & go on the same way...

$x_3 = x^3$ But remember feature scaling is more important because range of x^2 is square of x.

Normal Equation - Lec 4.6 & 4.7

- m training examples & n features.

$\Rightarrow \boxed{x_j^{(i)},\ y^{(i)}}$ $\quad i \to m$, $\quad j \to n$.

$\Rightarrow X\vec{\theta} =$ (training examples $\downarrow$, features $\rightarrow$)

$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & & \\ x_1^{(3)} & & & & & \\ \vdots & & & & & \\ x_1^{(m)} & & & & & \end{bmatrix}_{m \times n} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}_{n \times 1}$$

$$\Rightarrow \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

We want $\boxed{X\theta = y}$ Need to find θ to get the best fit.

$\Rightarrow$ Find the projection of $\vec{y}$ onto column space of X.

→ We are trying to transform θ using X to land on y but y must lie in $C(X)$ for this to happen. So we project it & settle for an approximate $\hat{\theta}$.

$$\Rightarrow X^T X \hat{\theta} = X^T y$$

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

→ projection error

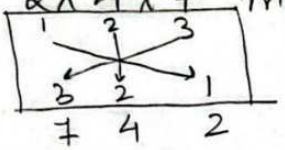
$\|e\|^2 = \|y - X\hat{\theta}\|^2$ is minimum.

$X^+ = (X^T X)^{-1} X^T$ is the pseudoinverse of X. (technically left inverse)

MATLAB

- pseudo inverse : pinv(A)
- transpose : B = A.'
- conj. transpose : B = A'
- permuting rows & columns : B = permute(A, [3, 2, 1])

If A was 2 x 4 x 7 matrix it is now 7 x 4 x 2.



$\Rightarrow \hat{\theta} = pinv(X'*X)*X'*y$

> Feature scaling is not needed for normal equation method.

Gradient descent	Normal Equation
:(→ Need to choose α	→ :) No need to choose α
:(→ Many iterations	→ :) No iterations.
:) → Works well even when m is large.	→ $\approx m^3$ is the cost of inverting $(X^TX)^{-1}$ ⇒ Slow for large data sets. $m \approx 1000$ is fine, ⇒ 1000 x 1000 $m \approx 10^5$ ⇒ use grad. descent.

→ When is X^TX non invertible? → use pseudo inverse.

or > linearly dep features

or > Too many features ⇒ $m \leq n$. ⇒ use regularization.

Lec 5.1 MATLAB / OCTAVE

> Commands

» ~= ≠

» && AND

» || OR

» disp(sprintf('6decimals : %.0.6f', a))

» format long.

» A = [12 ; 34 ; 56] => 3x2

» V = [1 ; 3 ; 7]

» V = 1:0.1:2 → 1x11 matrix.

» V = 1:6 → 1x6 matrix

» 2*ones(2,3) → 2 2 2
2 2 2

» zeros(1,3) → 0 0 0

» rand(1,3) → random.

» w = randn(1,3) → mean 0, st.dev 1
gaussian random

» hist(w) → histogram of w

» hist(w,50) → 50 bins => fine resolution.

» w = -6 + sqrt(10)*(randn(1, 10000)) => mean = -6
st.dev = $\sqrt{10}$
var = 10

» eye(4) => 4x4 identity.

» help eye.

>> size(A)

>> size(A,1) → no. of rows.
>> size(A,2) → no. of columns.
>> length(v) ⇒ longest dimension.

>> load featuresX.dat
load('featuresX.dat') } loads textfiles into MATLAB

? >> who → shows current variables.

? >> whos → detailed view → size, bytes, class

>> clear A → deletes a variable.

>> save hello.mat v; → saves variable v into a file hello.mat in binary format.

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

>> A(3,2) indexes to A_{32} element.

>> A(2,:) → every element in row 2

>> A([1 3],:) → get everything from first row & 3rd row

>> A(:,2) = [10; 11; 12] ⇒ second col. is replaced.

>> A = [A, [100; 101; 102]] ⇒ appends a column to the right.

>> A(:) ⇒ put all elements of A into a single col. vector. goes from 3×3 → 9×1

>> C = [A B] ⇒ $\begin{bmatrix} A & B \end{bmatrix}$

>> C = [A; B] ⇒ $\begin{bmatrix} A \\ B \end{bmatrix}$

Lec 5.3

» log (A) } log & exp of each element.
» exp (A)
» abs (v)

» v + ones (length (v), 1) → increments each value by 1
» v+1

» A' → conj transpose.

» max (a) ⇒ maximum value of a

» max (A) ⇒ column wise maximum.

» [val, ind] = max (a) ⇒ value & index

» a < 3 ⇒ element wise comparison.

» find (a < 3) ⇒ tells you indices of elements less than 3.

» A = magic (3) ⇒ N×N magic square.
⇒ rows, cols & diags sum up to same no.

» [r, c] = find (A >= 7) ⇒ row & col. index of each element >= 7 in A.

» sum (a) ⇒ adds all elements.

» prod (a) ⇒ prod.

» floor (a) ⇒ rounded down.
ceil (a) ⇒ rounded up.

» max (rand (3), rand (3)) ⇒ element wise max value b/w 2 3×3 matrices is returned as a 3×3 matrix

» max (A, [], 1) ⇒ column wise max
↳ along columns. if it was 2, it is taken per row.

>> max (max (A)) => max val. in the whole of A.

>> or max (A (:))

>> Sum (A, 2) => row wise sum!

>> A .* eye(length(A)) → diagonal entries left.

>> flipud (A) => flip up, down

>> pinv (A) => inverse.

Lec 5.4 Plotting.

```
>> plot (t, y1)
>> hold on
>> plot (t, y2, 'r')        % 'r' → red.
>> xlabel ('time')
>> legend ('sin', 'cos')
>> title ('my plot').
>> print -dpng 'myPlot.png'
>> close => closes figure.
>> figure(3)
>> subplot (1, 2, 1);        % 1x2 grid; 1st element
>> axis ([0.5 1 -1 1])       % xrange yrange
>> clf => clears figure.
```

>> imagesc(A) => plots a colourful picture representing the matrix

>> imagesc(A), colorbar, colormap gray;

>> imagesc(magic(15)), colorbar, colormap gray;

Lec 5.5 Control statements

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

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

>> i = 1;
   while true
    v(i) = 999;
    j = i+1;
    if i == 6
       break; => breaks out of while loop.
    end; => for if
   end; => for while.

>> if v(1) == 1
       disp('the value is one');
   elseif v(1) == 2
       disp(...);
   else
       disp(...);
   end;
```

Other useful MATLAB commands

>> y = linspace(-5, 5, 7) => Creates a vector of size 1x7 with values equally spaced b/w [-5 & 5].

>> y = linspace(-5, 5, 7)'
↳ creates column vector.

>> M = csvread('csvlist.dat', 1, 0, [1, 0, 2, 2])

=> read from row 1 to 2 & column 0 to 2.

>> permute(matrix, [2 1 3])

=> exchanges the rows & columns of a 3D matrix.

★ Functions.

> Create a file containing the function with the file name as the function name.m

Eg: squarethisnumber.m. {use wordpad not notepad}

File

```
function y = squarethisnumber(x)
{ y = x^2;
```

function body.

value to be returned.

argument it takes.

Command line

» squarethisnumber(5) ⇒ make sure you are in the right directory.

» addpath('C:\Users\mavarma\....')

↳ adds this path to the search path so files in this directory can be accessed even if we are in a different directory.

File.

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

} Can return multiple values.

Cmd

```
[a, b] = squareandcube(5);
```

Eg:-

```
>> X = [1 1 ; 1 2 ; 1 3]
>> y = [1 ; 2 ; 3]
>> theta = [0 ; 1]
```

File

```
function J = costFunctionJ(X, y, theta).
% x is the "design matrix" containing training examples.
% y is the "class labels"
m = size(X,1);          % no of training examples.
predictions = X*theta   % predictions of hypothesis on all m-examples.
sqrErrors = (predictions-y).^2;
J = 1/(2*m) * sum(sqrErrors);
```

```
>> j = costFunctionJ(X,y,theta).
```

Lec 5.6 Using Vectorized implementation.

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \longrightarrow \text{unvectorized.}$$

$$= \theta^T x \longrightarrow \text{vectorized.}$$

> Remember matlab index starts from 1.

>> prediction = theta' * x;

- Gradient descent

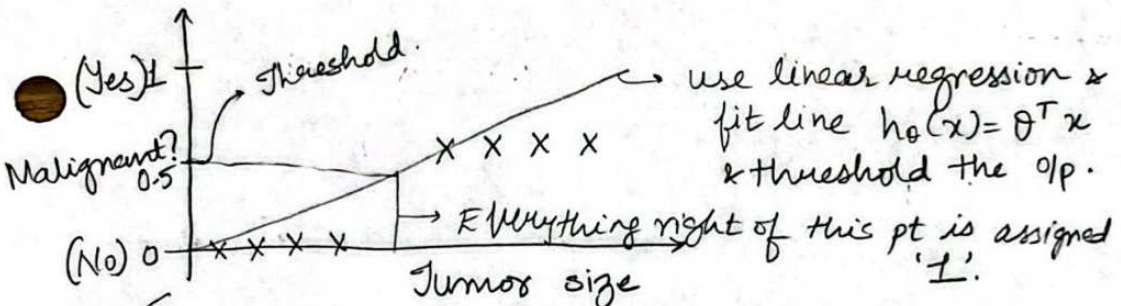
$$(h_\theta(x^{(1)}) - y^{(1)})x^{(1)} + (h_\theta(x^{(2)}) - y^{(2)})x^{(2)} + \dots$$

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

theta = theta - alph/m * x' * (h_theta(X) - y)

Lec 6.1 Logistic Regression → Classification problem.

$y \in \{0, 1\}$ 0: "Negative class"
1: "Positive class"



↳ This method doesn't work if we have outliers.

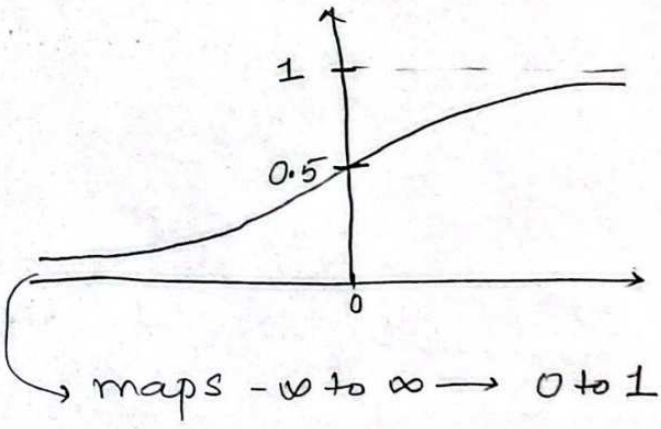
⇒ Develop an algorithm that has an output b/w $0 \le h_\theta(x) \le 1$

Lec 6.2 Logistic Regression.

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

$h_\theta(x) = g(\theta^T x)$ → modify the hypothesis

where $g(z) = \frac{1}{1+e^{-z}}$ → Sigmoid fn or Logistic fn



↳ maps $-\infty$ to ∞ → 0 to 1

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

> Find θs & fit to $h_\theta(x)$.

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

Eg: $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorsize} \end{bmatrix}$

if $h_\theta(x) = 0.7$

=> patient has 70% chance of malignant tumor.

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

> probability that $y=1$ given x, parameterized by θ.

> y must be 0 or 1

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

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

Lec 6.3 Decision Boundary.

Possible solution:

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

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

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

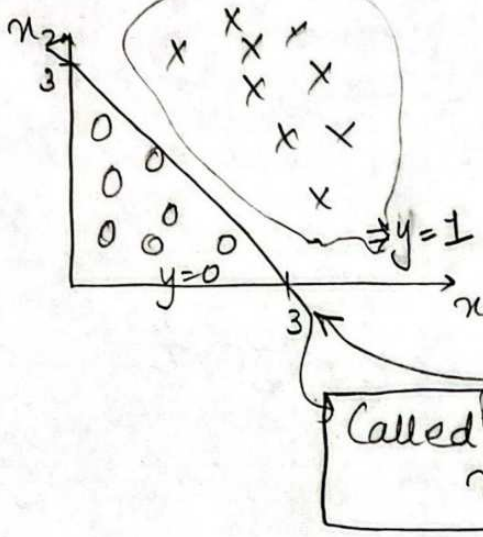
> $h_\theta(x) = g(\theta^T x)$ is ≥ 0 when $\boxed{\theta^T x \geq 0}$

$\Rightarrow$

$\boxed{y = 1 \text{ if } \theta^T x \text{ is } \geq 0}$

$\boxed{y = 0 \text{ if } \theta^T x \text{ is } < 0}$

Decision Boundary



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

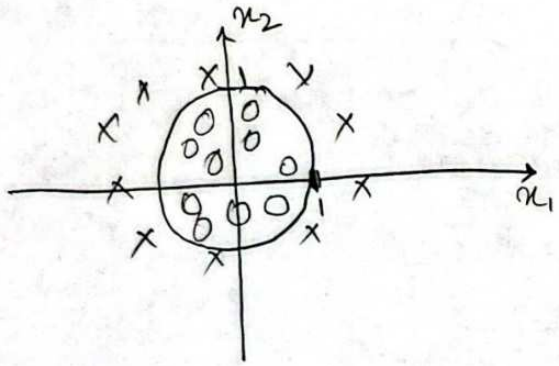
let $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$

$\Rightarrow$ predict $y = 1$ if $-3 + x_1 + x_2 \geq 0$.

$\Rightarrow x_1 + x_2 \geq 3$

Called Decision Boundary $x_1 + x_2 = 3$

> Nonlinear decision Boundary.



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

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

$\Rightarrow$ $x_1^2 + x_2^2 = 1$ $\rightarrow$ Decision boundary

> $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \ldots)$

Lec 6.4

given $X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \ldots & x_n^{(1)} \\ \vdots & & & & \\ x_0^{(m)} & & & & \end{bmatrix}$ & $y \in \{0, 1\}$

Recall earlier cost function. Modifying it a little:

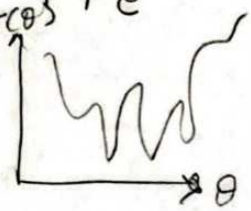
$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x^{(i)}, y))$

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

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

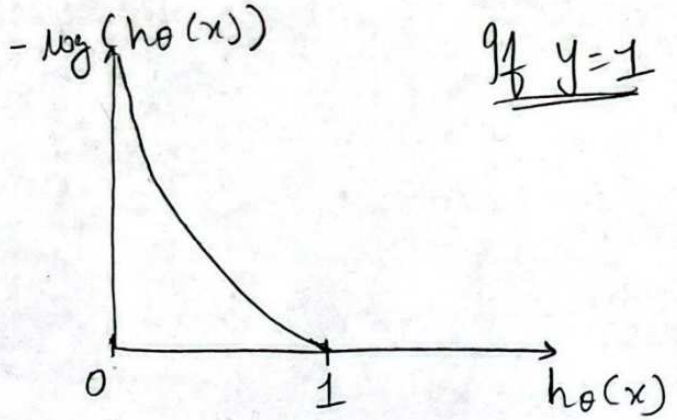
So far it's exactly the same J as linear regression.

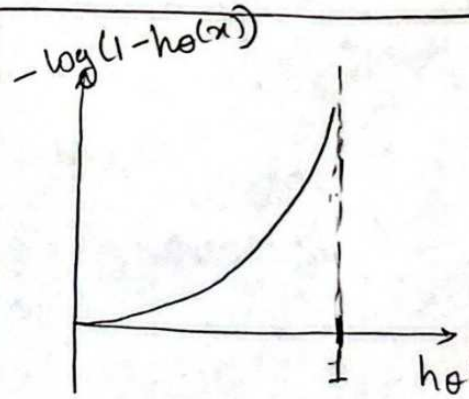
> Since now $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$, the resulting $J(\theta)$ vs θ is a nonconvex fct.



> Need to make it convex!

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





$h_\theta(x) = 1$ when $y = 0 \Rightarrow$ cost $\to \infty$

$h_\theta(x) = 0 \Rightarrow$ cost $= 0$!

$y = 1$ & $h_\theta(x) = 1 \Rightarrow$ cost $= 1$! Great!

$y = 1$ & $h_\theta(x) \to 0 \Rightarrow$ cost $\to \infty$! again Great!

→ The overall cost function is convex! → Not proved here though.

Logistic regression cost function.

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

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

more succint.

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

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

- → Derived from the principle of maximum likelihood estimation

To fit parameters θ:

- $\min_\theta J(\theta)$ → get θ.
- For a new x: output $h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$
- Now we minimize $J(\theta)$ using gradient descent!

Repeat {

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

}

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

> Algorithm look identical to linear regression.
> However definition of $h_\theta(x)$ is different.
> Feature scaling should make this better too.

Lec 6.6 Advanced Optimization

→ Given $J(\theta)$.
→ Find $\frac{\partial}{\partial \theta_j} J(\theta)$.

Then we use gradient descent,

> But we would use :- Conjugate descent, BFGS, L-BFGS.

Advantages

> No need to pick α. → There's an inner loop to compute α dynamically.
> Often faster than gradient descent

Disadvantages

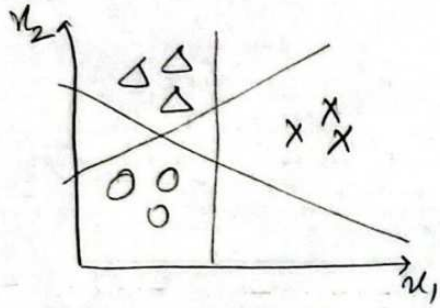
> More complex.

→ We could simply use a software built in library in MATLAB.
> Second half of the lecture gives some code to implement this stuff.

Lec 6.7 Multiclass Classification.

Eg:

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



Idea: One vs. All classification.

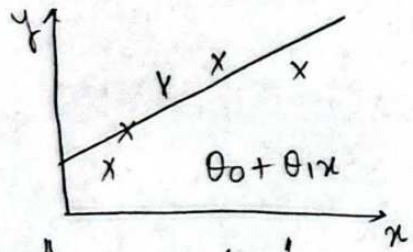
Class 1: △
Class 2: O
Class 3: X

> Create a new training set by combining n-1 classes

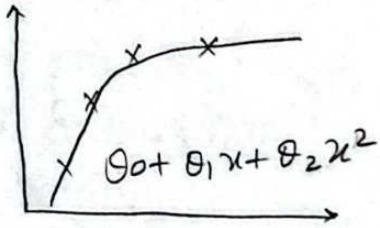
⇒ Class 1 vs. Class 2&3
2 vs 1&3
3 vs 1&2

} Solve these 3 binary logistic regression problem

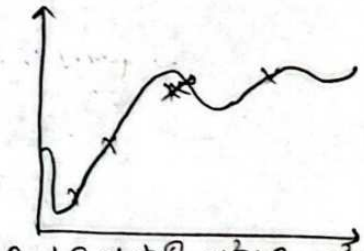
Lec 7.1 The problem of overfitting.



"Underfit" or "High Bias"



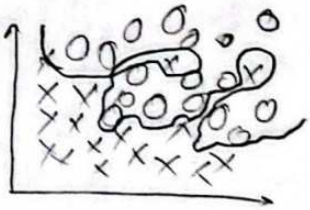
"Good fit"



"Overfitting" or "High variance"

> Overfitting: Too many features & cost fn → 0 but fail to generalize new examples.

> This can also occur in logistic regression.



→ Addressing overfitting.

> Plotting does not help when no. of features is high.

Options:

1. Reduce number of features.
   - Manually select which features to keep.
   - Model selection algorithm.

   ∴ - Reduces information available.

2. Regularization.
   - Keep all the features, but reduce magnitude of parameters θ_j.
   - Works well with large no. of features.

Lec 7.2 Regularization: Cost function.

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

$\Rightarrow \theta_3 \approx 0$ & $\theta_4 \approx 0$.

> Smaller values for parameters $\theta_0, \theta_1, \ldots, \theta_n$
   - "Simpler" hypothesis.
   - Less prone to overfitting.

> Modify cost function to shrink all the parameters, except θ_0.

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

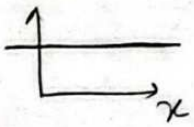
λ → Regularization parameter.

Controls the trade-off between a complex fit vs. regularized fit.

→ If $\lambda = 10^{10}$ or too large.

⇒ It would penalize $\theta_1 \dots \theta_n$ too much & $h_\theta(x) = \theta_0$.

⇒ It would underfit.



Lec 7.3 Regularized Linear regression.

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

$\min_\theta J(\theta)$

⇒ Repeat {

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

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

where $j = 1, 2, 3 \dots n$.

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

}.

$$\theta_j = \theta_j\underbrace{\left(1 - \alpha\frac{\lambda}{m}\right)}_{A} - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

→ $\left(1 - \frac{\alpha\lambda}{m}\right) < 1$ usually around 0.99. ⇒ θ_j becomes a bit smaller & is then updated by the second term

Normal Equation.

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m.$$

$m \times (n+1)$

$\min_\theta J(\theta)$

Now,

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

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

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

Non invertibility.

Suppose $m \le n$, $\Rightarrow$ $X^T X$ is singular.

$\rightarrow$ However if $\lambda > 0$, $\left(X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots \end{bmatrix} \right)$ becomes invertible!

Lec 7.4: Logistic regression.

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

$$\underbrace{+ \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2}_{\text{update.}}$$

Repeat {

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

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

$j = 1, 2, 3 \ldots n.$

}.

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

→ He briefly goes over how to implement regularization for advanced optimization algorithms.

Neural Networks.

Lec 8.1: Nonlinear Hypothesis

> If $n = 100$ ⇒ Taking all quadratic features would give $\frac{n(n+1)}{2}$ terms $\simeq x_1^2, x_1x_2, x_1x_3 \ldots x_2^2 + x_2x_3 + \ldots + x_{100}^2$

⇒ Too many features ⇒ overfitting / too much computation.

> Could reduce the no. of features.

> Cubic ⇒ on the order of n^3 ⇒ too large feature space.

> When feature size is high → use neural networks

Lec 8.2: Neurons and the brain.

- Computationally expensive.

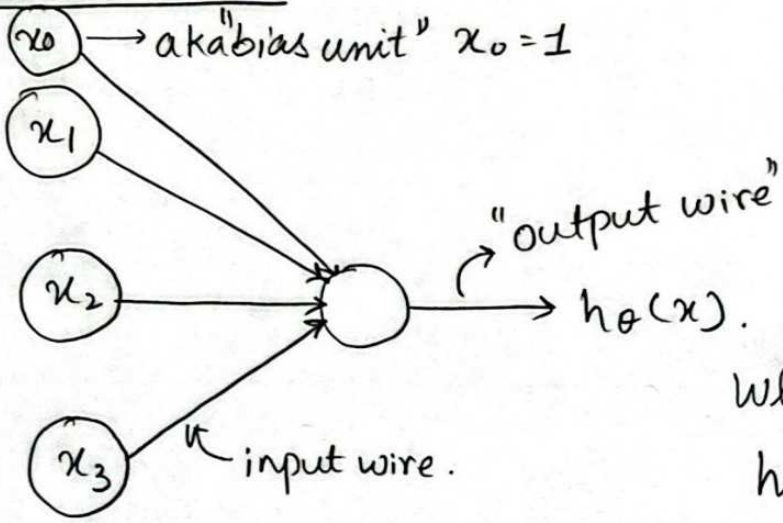
> "One learning algorithm"

> Auditory cortex learns to see when the connection from ear to brain is replaced by eye-brain.

> Neurorewiring experiments.

Lec 8.3 Model Representation.

Neuron model.



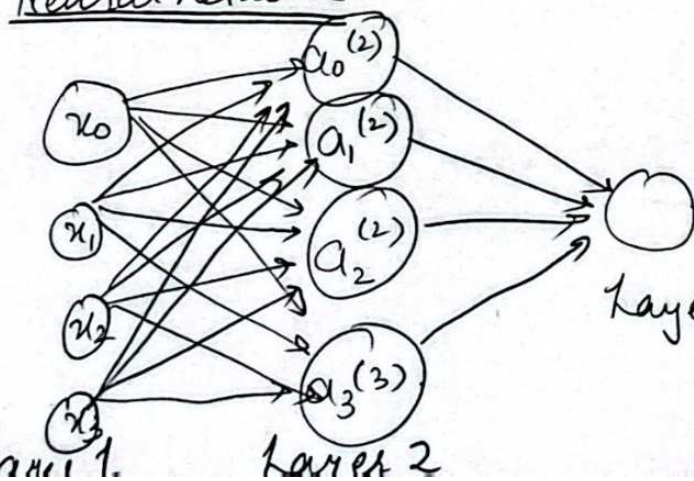
Where,

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

→ In this case the "activation function" is the sigmoid function $\frac{1}{1+e^{-z}}$.

→ θs are called "parameters" or "weights"

Neural network.



Layer 1 = Input layer

Layer 2 = Hidden layer

Layer 3 = Output layer.

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

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

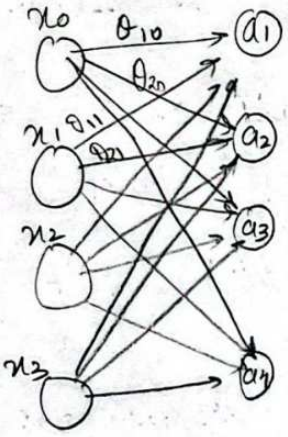
$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$

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

$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \dots + \theta_{13}^{(2)} a_3^{(2)})$$

→ If network has s_j units in layer j, s_{j+1} units in layer $j+1$ then $\theta^{(j)}$ will be of dimensions $s_{j+1} \times (s_j + 1)$. ↳ due to x_0.

Eg:-



layer 1 has 3 units
layer 2 has 4 units.

⇒ $\theta^{(j)}$ is $\theta^{(1)} = (4 \times 4)$.

$s_j + 1$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \theta_{10} & \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{20} & \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{30} & \theta_{31} & \theta_{32} & \theta_{33} \\ \theta_{40} & \theta_{41} & \theta_{42} & \theta_{43} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

s_{j+1} → [each neuron]

θ_{ab} : output ← a, input ← b

Lec 8.4. Contd...

let $z_1^{(2)} = \theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3$

$\Rightarrow a_1^{(2)} = g(z_1^{(2)})$

$z_1^{(2)} \rightarrow$ values associated with layer 2.

$\Rightarrow z_3^{(2)} = \theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3$

Vectorizing.

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(3)} \end{bmatrix} \qquad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$\Rightarrow z^{(2)} = \theta^{(1)} x$. where $z^{(2)}$ is a 3D vector!

$a^{(2)} = g(z^{(2)})$ $\hookrightarrow \mathbb{R}^3$

$\mathbb{R}^3$ ✓

$\rightarrow$ Using $a^{(1)} = x$ for consistency.

$$\boxed{\begin{aligned} z^{(2)} &= \theta^{(1)} a^{(1)} \\ a^{(2)} &= g(z^{(2)}) \end{aligned}}$$ * add $a_0^{(2)} = 1$

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

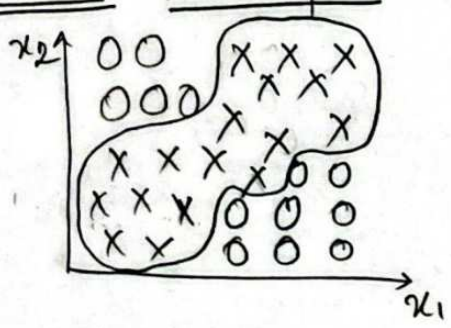
$\rightarrow$ This process is called forward propagation.

> With this view it just looks like a scaled version of logistic regression.

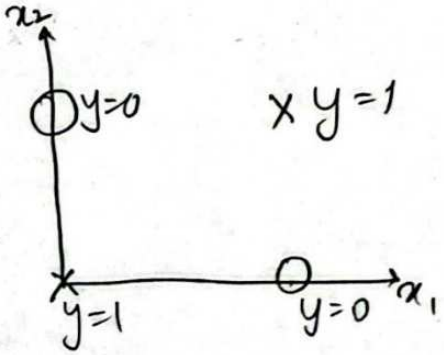
- The hidden layers are newly computed features.

> This takes care of the polynomial combinations?

Lec 8.5 Examples.



simple case.



$y \overset{?}{=} x_1$ XOR x_2

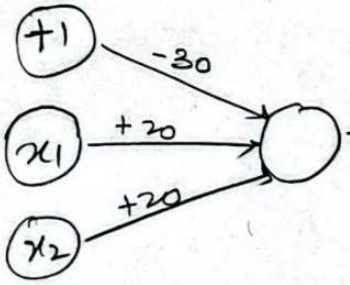
$y = x_1$ XNOR x_2

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

AND function.

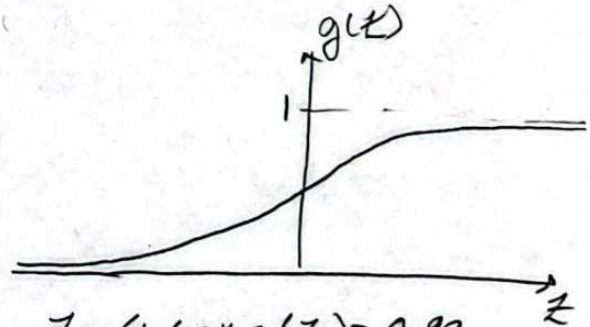
$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND x_2



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

$\theta_0^{(1)}$ $\theta_1^{(1)}$ $\theta_2^{(1)}$



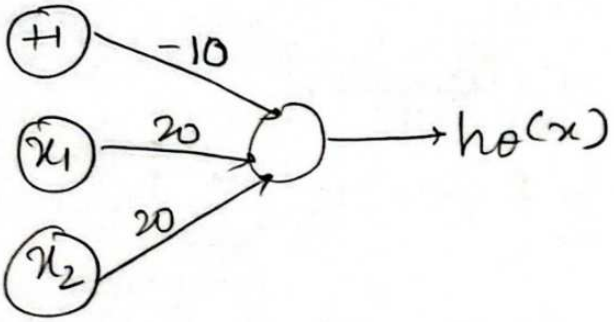
x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$z = 4.6 \Rightarrow g(z) = 0.99$

$z = -4.6 \Rightarrow g(z) = 0.01$

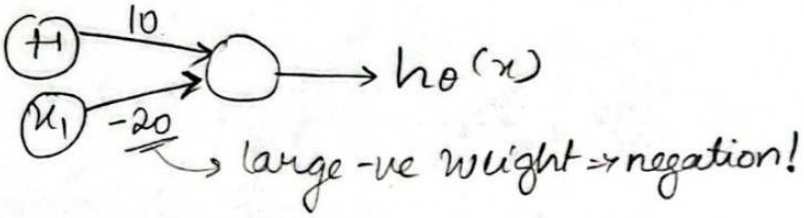
$\Rightarrow$ This computes $\approx x_1$ AND x_2

OR function.

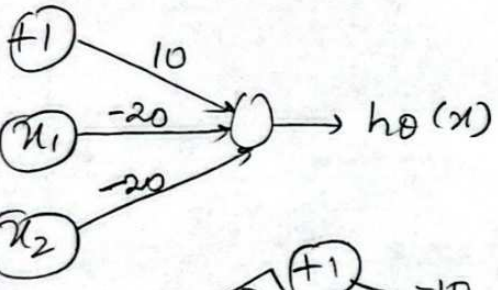


Lec 8.6 Contd...

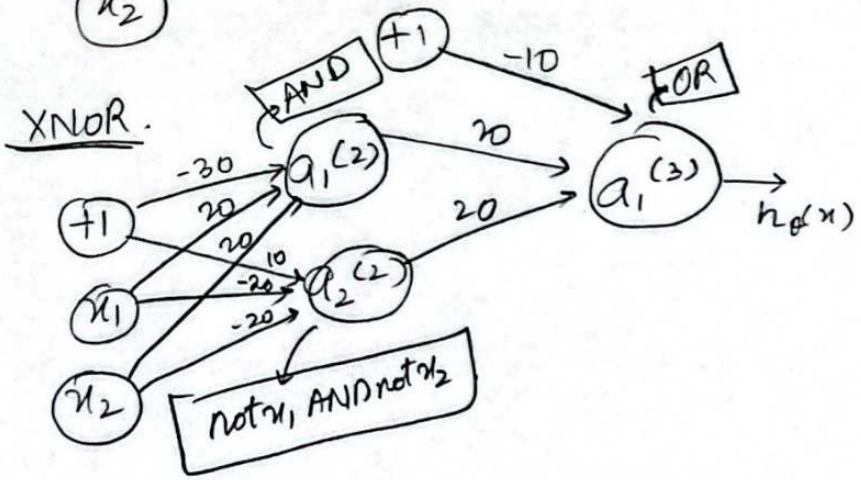
Negation.



(Not x_1) AND (not x_2)



XNOR.



x_1	x_2	a_1	a_2	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	

Lec 8.7 Multiclass classification.

→ Extension of one vs. all method.

→ output would be a vector of 4 units.

$$h_\theta(x) = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

⇒ 4 logistic regression classifiers.

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

$y^{(i)}$ ↑ vector $\mathbb{R}^4$

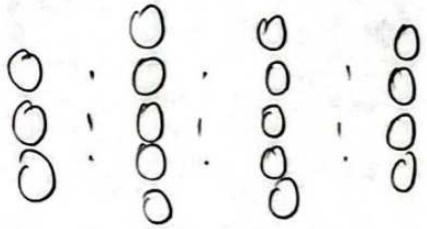
→ $h_\theta(x^{(i)}) \approx y^{(i)}$

Lec 9.1 : Cost function.

L = total no. of layers.

s_l = no. of units in layer l.

s_L = index of final layer = s_4.



Binary Classification

$y = 0$ or 1

$h_\theta(x) \in \mathbb{R}$.

$s_L = 1$

$K = 1$.

where $y \in \mathbb{R}^K$

Multiclass Classification.

$y \in \mathbb{R}^4$

K output units.

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

$s_L = K$.

$K \geq 3 \Rightarrow$ use one vs. all method.

Cost function.

→ Generalizing from logres..

$h_\theta(x) \in \mathbb{R}^K \qquad (h_\theta(x))_i = i^{th}$ output.

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

Lec 9.2 Backpropagation Algorithm

> An algorithm to minimize the cost function.

Need to compute: $J(\theta)$ & $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$.

Given one training example (x, y):

Apply forward propagation.

$a^{(1)} = x.$

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

$a^{(2)} = g(z^{(2)})$ & add $a_0^{(2)}$

repeat ↓

$z^{(3)}$

↓

$a^{(3)}$

↓

$z^{(4)}$

↓

$$\boxed{a^{(4)} = h_\theta(x) = g(z^{(4)})}$$

→ For derivatives use backpropagation.

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

- For each output unit ($L=4$).

$$\delta_j^{(4)} = a_j^{(4)} - y_j \longrightarrow \text{could use a vectorized implementation}$$
$$\hookrightarrow (h_\theta(x))_j$$

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \longrightarrow \text{derivative!}$$
$$= a^{(3)} .* (1 - a^{(3)})$$
$$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$
$$= a^{(2)} .* (1 - a^{(2)})$$

→ Not proved here but the result is:

- $$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)}$$ → {here we are ignoring λ ⇒ $\lambda = 0$}

Backpropagation algorithm.

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

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

(Capital Δ.)

For $i = 1$ to m

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

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

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

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

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

Leave the loop...

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

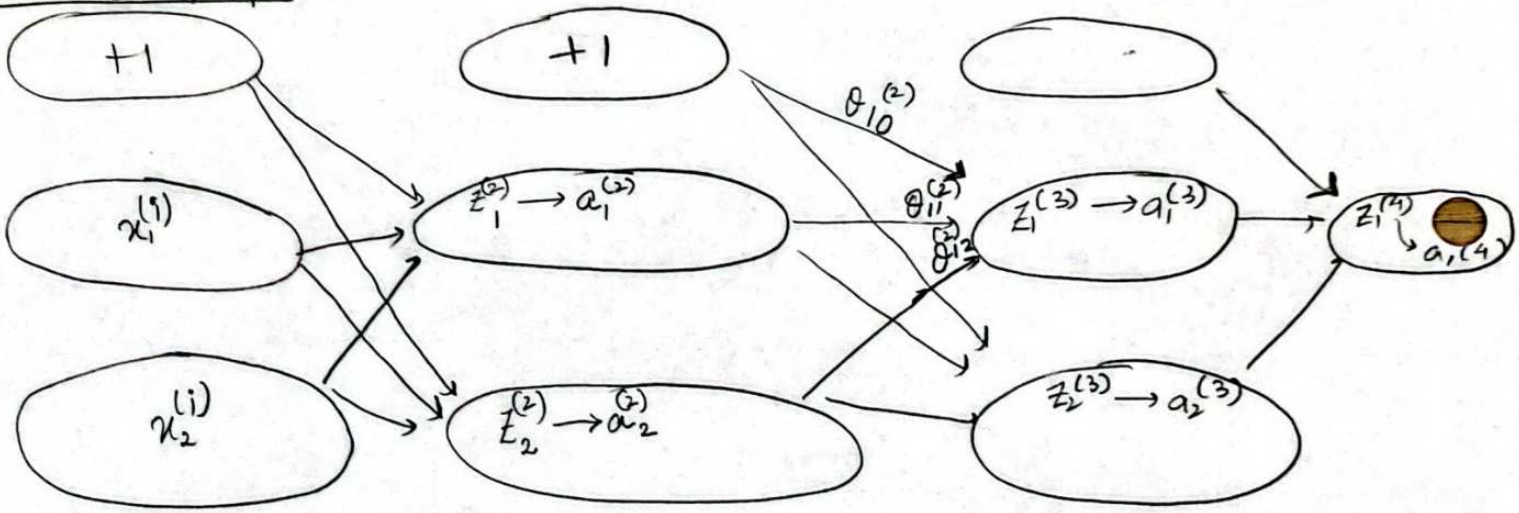
$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

& finally,

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = D_{ij}^{(l)}$$

Lec 9.3 Backpropagation Intuition contd...

Forward prop



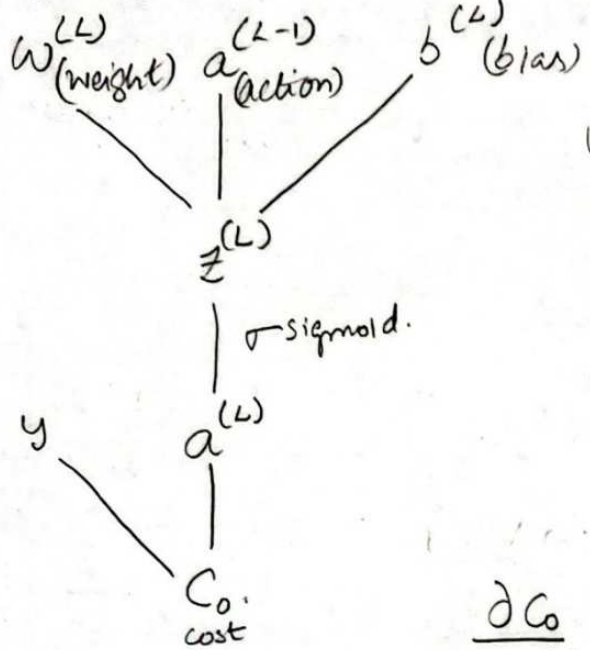
$$z_1^{(3)} = \theta_{10}^{(2)} x1 + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_1^{(2)}$$

→ δ is the partial derivative $\left(\frac{\partial}{\partial z_j^{(l)}} cost(i)\right)$ of the intermediate terms. They are a measure of how much the weights must be changed.

→ It is like a distributed feedback.

> Backpropagation by 3Blue1Brown.

> Eg: of 1 neuron per layer ○—○—○—○.



We want $\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}}$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$C_0 = (a^{(L)} - y)^2$$

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y).$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$ → previous neuron strength determines how much the weight affects $z^{(L)}$.

$$\Rightarrow \frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$$ → averaging deriv. over all training examples.

→ We also need $\frac{\partial C_0}{\partial b^{(L)}} = 1 \cdot \sigma'(z^{(L)}) 2(a^{(L)} - y).$

$$\frac{\partial C_0}{\partial a^{(L-1)}} = w^{(L)} \sigma'(z^{(L)}) 2(a^{(L)} - y).$$

→ For multiple neurons, C_0 is now a sum of all neurons.

Lec 9.4 Unrolling parameters.

> Unrolling matrices into vectors to use inbuilt functions in MATLAB.

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

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

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

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

→ To go back

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

→ Pulls out first 110 elements.

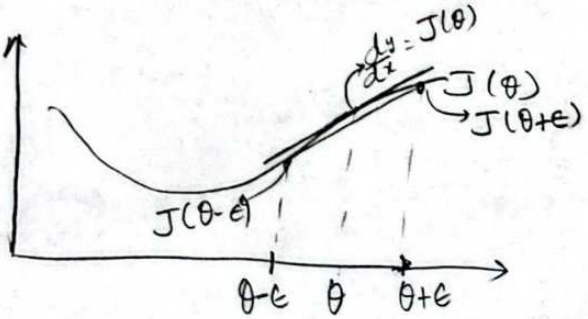
← puts them in 10 x 11 matrix.

→ fminunc is the function where we use this vector.

→ This is more for the advanced algorithms.

Lec 9.5 Gradient Checking.

> Eliminates subtle bugs in backprop. that are not obvious.



We approximate $\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}$

Where, $\epsilon \simeq 10^{-4}$

↑ 2 sided difference

```
>gradApprox = (J(theta+epsi) - J(theta-epsi))/(2*epsi).
```

> Let $\theta = [\theta_1, \theta_2 \ldots \theta_n] \in \mathbb{R}^n$.

Then,

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

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

$$\vdots$$

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

⇒ Numerically computing the derivative

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

↳ $\frac{\partial}{\partial i} J(\theta)$.

Implementation Note

1) Implement backprop to compute DVec
2) Implement numerical gradient check to compute gradApprox
3) Make sure they give similar values.
4) Turn off grad check. Use backprop code for learning.
   ↳ If you do not turn off, code will be very slow.

Lec 9.6 - Random Initialization

→ For gradient descent & advanced optimization method we need to initialize θ.

→ initialTheta = zeros(n,1)? → Does not work for neural networks. $\Rightarrow a_1^{(2)} = a_2^{(2)}$ & $\delta_1^{(2)} = \delta_2^{(2)}$ & $\frac{\partial}{\partial \Theta_{01}^{(1)}} J = \frac{\partial}{\partial \Theta_{02}^{(1)}} J$ => not going to work.

aka: problem of symmetric weights.

Sol: break symmetry using Θ_{ij} as a random value b/w $-\epsilon, \epsilon$,

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

(10,11): 10x11 matrix

↳ different ϵ from gradient checking ϵ.

Lec 9.7 Putting it together.

Step(1): Pick an architecture.

no. of i/p units = no. of features

no of o/p units = no. of op classes.

hidden layers → use same no. of units in each hidden layer.

→ No of hidden units => more is better!

Step(2): Randomly initialize weights.

Step(3): Implement forward prop to get $h_\theta(x)$ for any $x^{(i)}$

Step(4): Implement code to compute cost fn $J(\Theta)$

Step(5): Implement backprop to compute partial derives. $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.

↳ Use for loop. more advanced usage can be done using vectorization but not recommended for beginners.

Step(6): Use grad check to compare the partial deriv. terms & then disable it

Step(7): Use grad descent or adv. optimization with back prop to minimize $J(\Theta)$ as a fn of Θ.

→ Lec 10.1 Advice for applying Machine Learning

> What to do if new data set doesn't work? → Regularized linear regression.

- Get more training examples
- Try smaller set of features
- Try getting additional features
- Try polynomial features.
- Decrease or increase λ

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

But what to choose?

Machine Learning Diagnostic.

> Diagnostics can be slow but very useful.

> How do we diagnose overfitting for large number of features?

→ Split data into training set and test set 70% & 30%.

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

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

> Ideally distribute the data randomly b/w training & test set.

Lec 10.2

Training/testing procedure.

Linear regression:
- Learn parameter θ from training data. (minimizing $J(\theta)$)
- Compute test set error:

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

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).

Alternative to computing test set error.

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

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

↳ Fraction of data in test set that the hypothesis has mislabelled

Lec 10.3 Model Selection

- What kind of polynomial is best fit? To avoid overfitting.
- d = degree of polynomial to pick.

1. $h_\theta(x) = \theta_0 + \theta_1 x \longrightarrow \theta^{(1)}$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \longrightarrow \theta^{(2)}$
3. ⋮
10. $h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_{10} x^{10} \longrightarrow \theta^{(10)}$

θ vectors generated for each hypothesis.

- For each degree hypothesis find J_{test} test set error. Then select the $h_\theta(x)$ which has the least test set error.

> But this may not be the best hypothesis for a generalised -ised model. Since it is chosen to fit the test set very well it may not be good for other data → overfitting the test set.

Solution?

> Split data in 3 parts → training set, cross validation set (CV), test set → 60%, 20%, 20%

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

$x_{cv}^{(1)}$ $y_{cv}^{(1)}$
$x_{cv}^{(2)}$ $y_{cv}^{(2)}$ $\left(x_{cv}^{(i)}, y_{cv}^{(i)}\right)$
$x_{cv}^{(m_{cv})}$ $y_{cv}^{(m_{cv})}$

$x_{test}^{(1)}$ $y_{test}^{(1)}$
$x_{test}^{(2)}$ $y_{test}^{(2)}$ $\left(x_{test}^{(i)}, y_{test}^{(i)}\right)$
$x_{test}^{(m_{test})}$ $y_{test}^{(m_{test})}$

Train/validation/test error

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

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

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

Model selection.

1) $h_\theta(x) = \theta_0 + \theta_1 x \longrightarrow \min_\theta J(\theta) \longrightarrow \theta^{(1)} \longrightarrow J_{cv}(\theta^{(1)})$
2) $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \longrightarrow \min_\theta J(\theta) \longrightarrow \theta^{(2)} \longrightarrow J_{cv}(\theta^{(2)})$
⋮
10) $h_\theta(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10} \longrightarrow \min_\theta J(\theta) \longrightarrow \theta^{(10)} \longrightarrow J_{cv}(\theta^{(10)})$

Then pick the hypothesis $h_\theta(x)$ with lowest $J_{cv}(\theta)$.

→ Pick $\theta_0 + \theta_1 x_1 + \ldots + \theta_4 x^4$ if $J_{cv}(\theta^{(4)})$ was least.

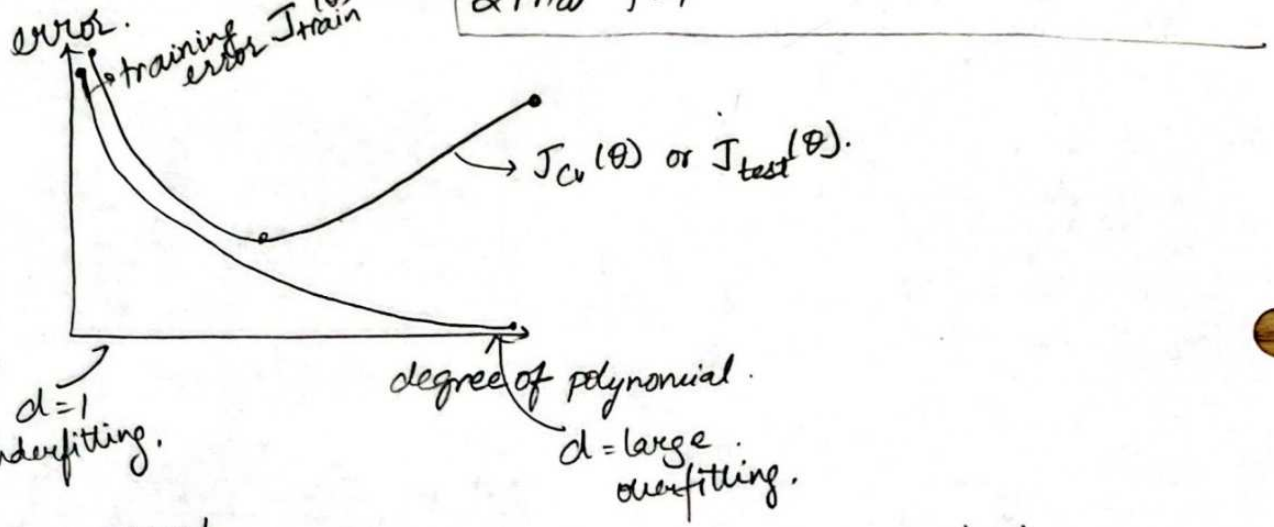
→ Estimate generalization error for test set $J_{test}(\theta^{(4)})$.

Lec 10.4

Diagnosing bias vs. variance.
(Underfitting) (overfitting)

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

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



> High $J_{train} \overset{\text{High}}{\approx} J_{cv}$ => high bias problem => increase d.

> High J_{cv} & Low J_{train} => high variance problem => decrease d.

$J_{cv} \gg J_{train}$

Lec 10.5 **Regularization and Bias/Variance.**

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

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

→ λ is large ⇒ underfit since $\theta_1, \theta_2 \ldots \approx 0$. ⇒ High bias.

→ λ is small ⇒ overfit.

→ How to choose λ?

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

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

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

} Do not include regularization.

→ Try $\lambda = 0, 0.01, 0.02, 0.04, 0.08 \ldots, 10.24$ ⇒ we get 12 models.

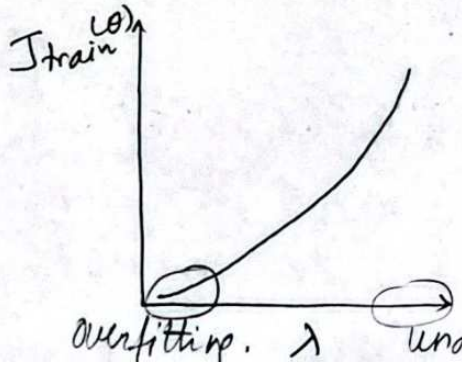
→ 1) $\lambda = 0 \longrightarrow \min_\theta J(\theta) \rightarrow \theta^{(1)} \longrightarrow J_{cv}(\theta^{(1)})$

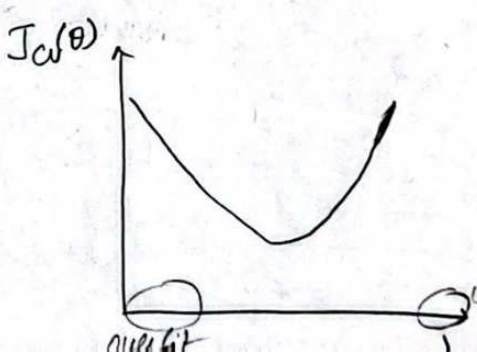
2) $\lambda = 0.01 \longrightarrow \min_\theta J(\theta) \rightarrow \theta^{(2)} \longrightarrow J_{cv}(\theta^{(2)})$

⋮ ⋮

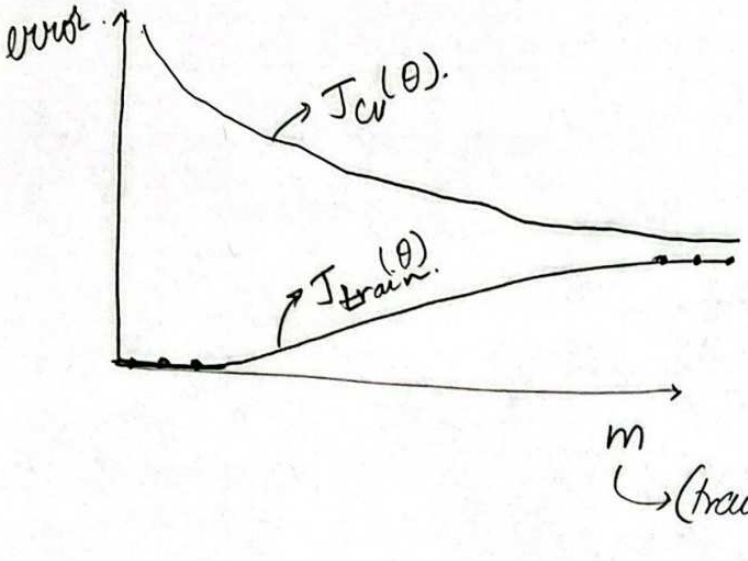
12) $\lambda = 10.24 \longrightarrow \min_\theta J(\theta) \rightarrow \theta^{(12)} \longrightarrow J_{cv}(\theta^{(12)})$

→ Pick say $\theta^{(5)}$ if $J_{cv}(\theta^{(5)})$ is lowest. Use $\theta^{(5)}$ to test $J_{test}(\theta^{(5)})$





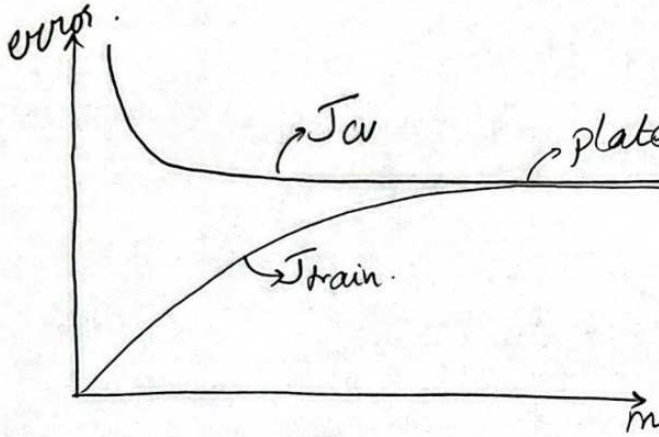
Lec 10.6 Learning Curves.



→ Deliberately reduce training set size & observe error.

→ If $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \Rightarrow$ for $m = 1, 2, 3$ J_{train} would be 0.

High bias case.

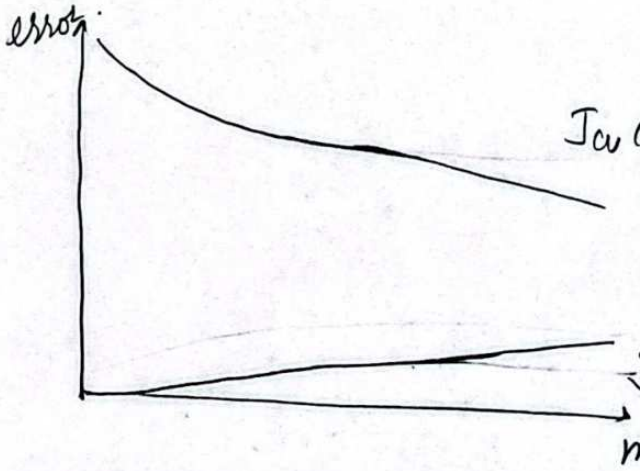


plateau occurs since best possible straight line doesn't change as m↑

→ high bias ⇒ $J_{cv} \simeq J_{train}$ at high m.

→ If we have high bias ⇒ getting more training data is not going to help. → Don't waste time collecting training data.

High variance.



Large gap b/w J_{cv} & J_{train}.

→ But as m↑ $J_{cv}(\theta)$ keeps ↓

⇒ more training data helps!

- Get more training examples → fixes high variance.
- Try smaller sets of features → fixes high variance.
- Try getting additional features → fixes high bias (not always though)
- Try adding polynomial features ↗
- → Decreasing λ → fixes high bias
- → Increasing λ → fixes high variance.

Neural networks

→ "Small" neural network ⇒ few parameters ⇒ prone to underfitting.

→ "Large" NN ⇒ computationally expensive ⇒ prone to overfitting. & need to use regularization to correct.

→ Large NN with regularization is often a better approach.

→ No. of hidden layers? Check for 1, 2, 3 layers & compute J_{CV} & J_{test} to see which is best.

c 11.1 Machine learning Systems Design.

Building a spam classifier.

features: deal, buy, discount, now...

$x_j = \begin{cases} 1 & \text{if a word appears.} \\ 0 & \text{otherwise.} \end{cases}$

$$x = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \text{deal} \\ \text{buy} \\ \text{discount} \\ \text{now} \\ \vdots \end{matrix}$$

* Could use sophisticated features based on email routing info.

Lec 11.2 Error analysis

> Start with a simple algorithm and test it on cross validation data
> Plot learning curves to decide if more data, more features etc.
> Error analysis: Manually examine the examples that your algorithm made errors on.
↳ Gives insight on new features needed.

Lec 11.3 Error metric for Skewed Classes.

Eg:- Cancer classification.

> Say we have 99% correct diagnoses ⇒ 1% error.
> But if only 0.5% patients have cancer this algorithm does worse than an algorithm that says nobody has cancer.
> # +ve or -ve examples are disproportionally large ⇒ Skewed classes.
⇒ Classification accuracy is not a good enough metric.
> Use Precision/Recall metric

Precision/Recall.

		Actual class: 1	Actual class: 0
Predicted class	1	True positive.	False positive.
	0	False negative.	True negative.

Precision

(Of all patients where we predicted $y=1$, what fraction actually has cancer?)

$$\frac{\text{True positives}}{\#\text{ predicted positives}} = \frac{\text{True positives}}{\text{True pos} + \text{False pos.}}$$

Recall

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

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

→ Both precision & recall must be high.

→ Recall = 0 if $y=0$ is predicted all the time.

↑ precision ⇒ positive predictions are mostly correct

↑ recall ⇒ Did not miss predicting a positive case.

Lec 11.4 Trading off precision and recall.

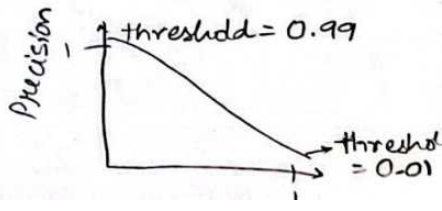
Logistic regression: $0 \leq h_\theta(x) \leq 1$

predict 1 if $h_\theta(x) \geq 0.5$

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

→ Could change threshold ⇒ predict 1 if $h_\theta(x) \geq 0.7$.

⇒ Higher precision and lower recall.



F1 score (F score)

How to compare precision/recall numbers?

- Average? NO!
- Use harmonic mean: $\frac{2PR}{P+R}$ (F score) or (F1 score) ⇒ makes sure both should be large.
- Fscore lies b/w 0 & 1.

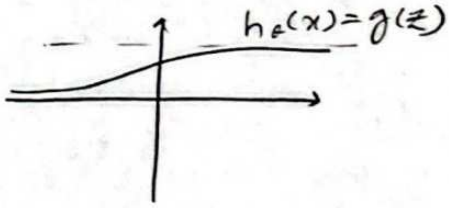
Lec 11.5 Data for Machine learning.

- "It's not who has the best algorithm that wins. It's who has the most data".
- When is this true?
- Useful test: Given the input x, can a human expert confidently predict y? If yes, more data helps.
- Very large training set ⇒ Unlikely to overfit even with large no of features and parameters
  - ⇒ many parameters ⇒ low bias.
    massive dataset ⇒ low variance

Lec 12.1 Support Vector Machines.

Alternate views of logistic regression.

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



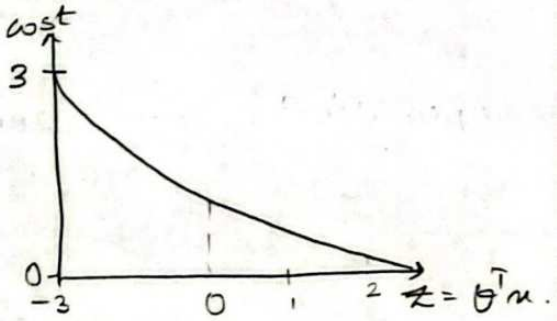
If $y = 1$, we want $h_\theta(x) \approx 1$, $\Rightarrow \theta^T x \gg 0$

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

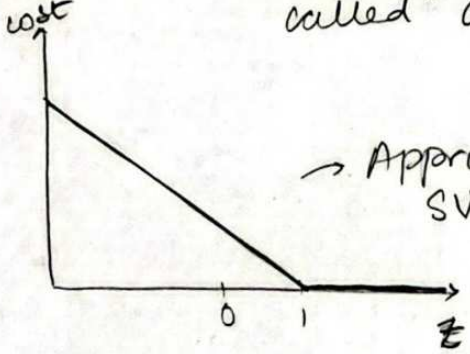
Cost of example: $= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y) \log \left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$

Case i: If $y = 1$ we want $\theta^T x \gg 0$ $\Rightarrow$ only 1st term matters.

$\Rightarrow$ cost $\simeq -\log \frac{1}{1+e^{-\theta^T x}}$

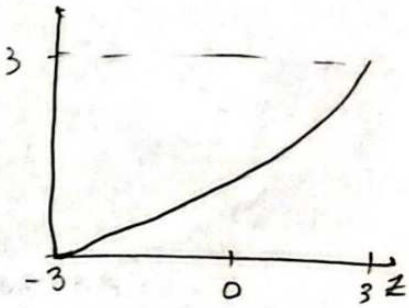


$\rightarrow$ We use a new cost function called $\text{cost}_1(z)$

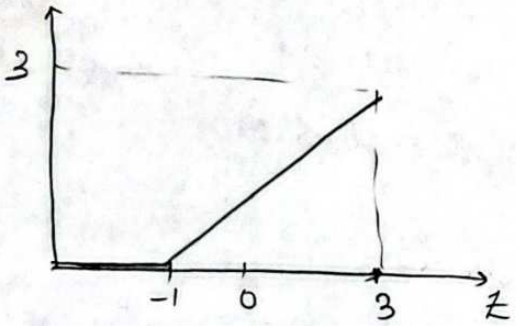


$\rightarrow$ Approximating it as 2 straight lines gives us the SVM $\Rightarrow$ computationally better.

Case ii: $y = 0$ $\Rightarrow$ cost $= -\log\left(1 - \frac{1}{1+e^{-z}}\right)$



$\rightarrow$ We use a new cost function called $\text{cost}_0(z)$



Logistic regression cost fu:-

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \underbrace{\left(-\log h_\theta(x^{(i)})\right)}_{\text{cost}_1(\theta^T x^{(i)})} + (1-y^{(i)}) \overbrace{\left(-\log(1-h_\theta(x^{(i)}))\right)}^{\text{cost}_0(\theta^T x^{(i)})}\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

SVM cost fu:

$$\Rightarrow \min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) + \frac{\lambda}{2m}\sum_{j=0}^{n} \theta_j^2\right]$$

Using a different convention.

> Remove $\frac{1}{m}$ $\Rightarrow$ doesn't change the optimum value.

> Instead of parameterizing $A + \lambda B$ (regularization parameter), we use $CA + B$.

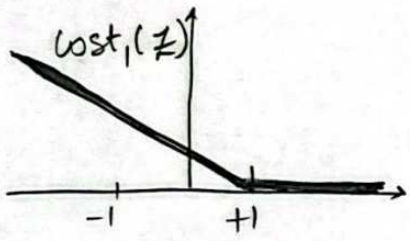
$$\min_{\theta} \cdot C \sum_{j=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

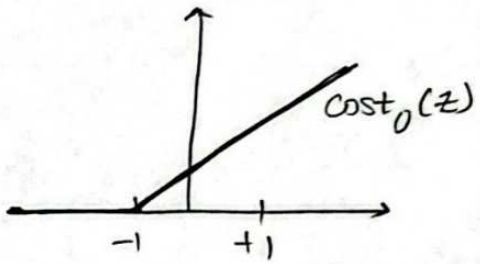
Hypothesis

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

SVM directly predicts 1 or 0.

Lec 12.2 Large Margin Intuition.





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

} Extra safety or margin for error.

SVM Decision Boundary.

If C is very large $\Rightarrow$ when $y^{(i)} = 1$, $\theta^T x^{(i)} \geq 1$

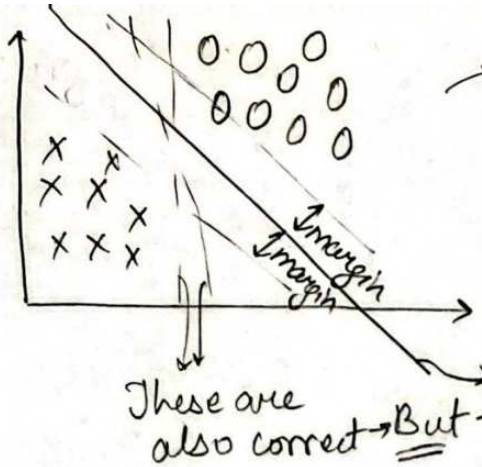
when $y^{(i)} = 0$, $\theta^T x^{(i)} \leq -1$

$$\min C \cdot 0 + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

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

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

} Solving this gives an interesting decision boundary.



→ linearly seperable case.

→ SVM chooses largest margin => ∴ it is aka large margin classifier.

→ Large margin classifiers are sensitive to outliers if C is large!

Lec 12.3 Math behind Large Margin Classification.

> Vector inner product.

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \Rightarrow u \cdot v = u^T v = p \cdot \|u\|$$

(p: projection of v on u; $\|u\| = \text{norm}(u) = \sqrt{u_1^2 + u_2^2}$)

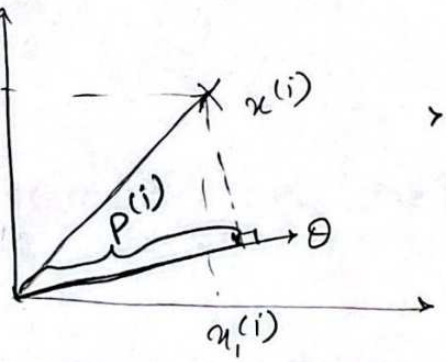
$$\Rightarrow u^T v = v^T u = p \cdot \|u\| = p \cdot \|v\| = u_1 v_1 + u_2 v_2$$

(p: v on u; p: u on v)

→ Going back => $\min_\theta \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 = \frac{1}{2}\left(\sqrt{\theta_1^2 + \theta_2^2}\right)^2 = \|\theta\|$

↳ assuming $\theta_0 = 0$ & $n = 2$.

→ $\theta^T x^{(i)} = p^{(i)} \|\theta\|$
$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}$



> Representing each data point as a vector. Both features & parameters.

SVM Decision Boundary

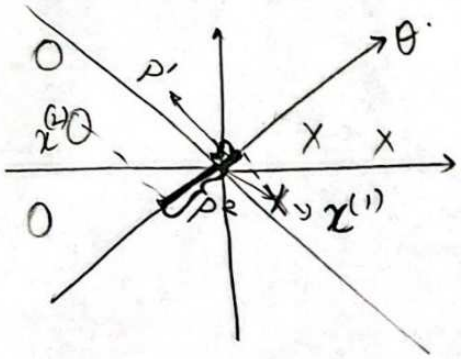
$$\min_{\theta} \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 = \frac{1}{2} \|\theta\|^2$$

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

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

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

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



- $\theta_0 = 0 \Rightarrow$ decision boundary goes through origin.
- θ vector is $\perp^{lar}$ to decision boundary.
  - ↳ can be proven

$p^{(1)} \|\theta\| \geq 1$ but we see $p^{(1)}$ is small

$\Rightarrow \|\theta\|$ must be large

$p^{(2)} \|\theta\| \leq -1$ but $p^{(2)}$ is small

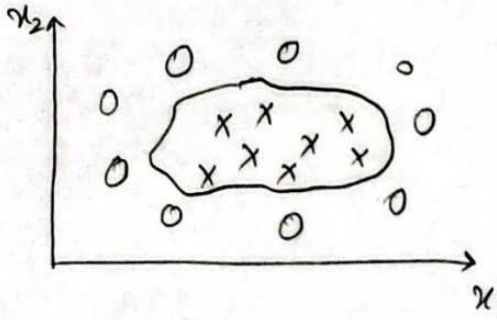
$\Rightarrow \|\theta\|$ must be large.

But we want $\|\theta\|$ to be small.

→ Therefore, the decision boundary moves around to get $\max p^{(i)}$ which comes out to having largest margins!

→ $p^{(i)}$ itself is the magnitude of the margin.

Kernels - I - SVM for nonlinear decision boundaries.



> Use complex features

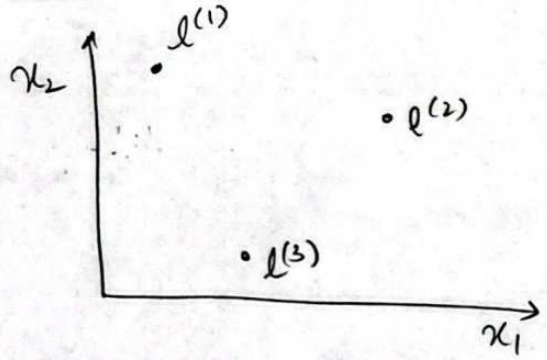
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \cdots$$

> Uses new notation: $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

$f_1 = x_1$, $f_2 = x_2$, $f_3 = x_1 x_3 \ldots$

> Is there a better choice than $f_1, f_2, f_3 \ldots$?

> **Kernel**



> Given x, compute feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$ that are manually chosen (for now).

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

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

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

↳ called a Gaussian Kernel.

> Similarity functions are called Kernels.

$\Rightarrow f_i = k(x, l^{(i)})$

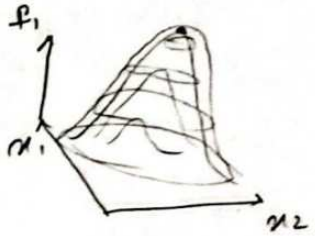
$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(\frac{-\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$

If $x \approx l^{(1)}$: $f_1 = \exp(-0) \approx 1$

If x is far from $l^{(1)}$ $\Rightarrow$ $f_1 = \exp(-\text{large}) \approx 0$.

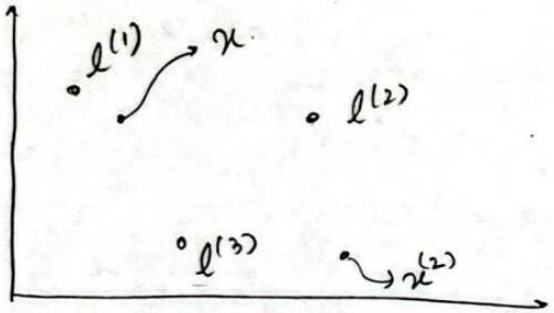
Example

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



$\rightarrow \sigma^2 = 0.5 \Rightarrow$ width of peak is narrower. $\rightarrow$ variance!

> Predict "1" when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$.



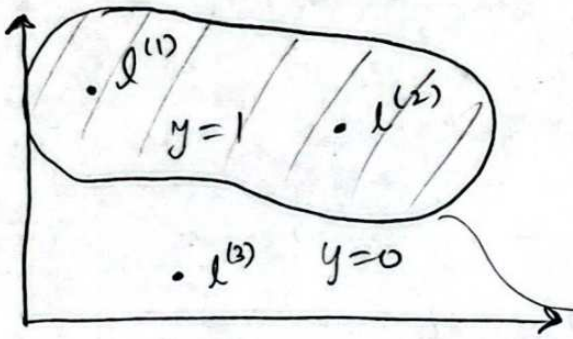
Say we end up with $\theta_0 = -0.5$
$\theta_1 = 1$
$\theta_2 = 1$
$\theta_3 = 0$

Given x, $f_1 = 1$, $f_2 = 0$, $f_3 = 0$.

$\Rightarrow \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3$
$= 0.5 \geq 0 \Rightarrow$ predict $y = 1$.

Given $x^{(2)} \Rightarrow f_1, f_2, f_3 \approx 0$.
$\Rightarrow \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \simeq -0.5$
$\Rightarrow$ predict $y = 0$.

$\Rightarrow$

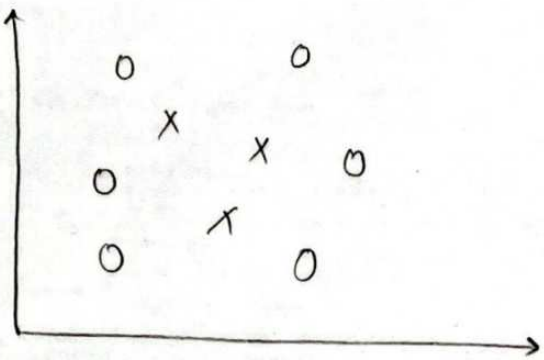


Points near $l^{(1)}$ & $l^{(2)}$ we predict positive!

> How do we choose landmarks? Which Kernel to use?

Lec 12.5 Kernels - II

How to choose landmarks?



→ Start with choosing landmarks as the training examples itself
⇒ we get m landmarks.

⇒ $l^{(1)} = x^{(1)}$, $l^{(2)} = x^{(2)}$ ... $l^{(m)} = x^{(m)}$

Given example x:

$f_0 = 1$

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

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

⇒ $f = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{bmatrix}$

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

$f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$

$f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$

$f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)})$ → similarity of $x^{(i)}$ & $l^{(i)} = x^{(i)}$ ⇒ = 1

$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$.

→ We had $x^{(i)} \in \mathbb{R}^{n+1}$, now we can use a new feature vector

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

SVM with kernels

- Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$
- Predict "$y=1$" if $\theta^T f \geq 0$ $\quad \theta \in \mathbb{R}^{m+1}$
- Training: SVM, use f instead of x.

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

↳ θ_0 is not regularized

- If we ignore θ_0 $\Rightarrow$ $\sum_{j=1}^{m} \theta_j^2 = \theta^T \theta$.
  - $\Rightarrow$ Some SVMs use $\theta^T M \theta$ instead of $\sum \theta_j^2$ where M is a matrix that depends on the kernel. It minimizes something similar to $\|\theta\|^2$. Makes it more efficient for large m.
- Kernels don't work very well with logistic Regression.

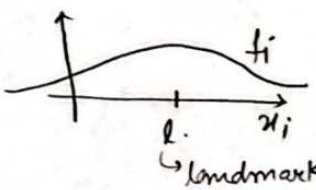
→ Choosing 'C'

Large C $\Rightarrow$ lower bias, high variance. → same as small

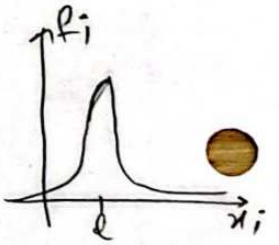
Small C $\Rightarrow$ Higher bias, low variance. ↳ underfit.

→ Choosing σ^2

Large σ^2 $\Rightarrow$ Features f_i vary more smoothly. Higher bias, lower variance



Small σ^2 $\Rightarrow$ Features f_i vary less smoothly. ↓bias, ↑ variance.



Sec 12.6 Using an SVM

> Use SVM software packages (liblinear, libsvm, ...) - libraries

Need to specify:

> Choice of parameter C.

> Choice of Kernel (similarity function):

Eg ① No kernel or linear kernel

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

n: large, m small.

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

② Gaussian Kernel.

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

Need to choose σ^2.

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

> MATLAB will ask to provide a function where Kernel is defined.

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

function f_i = Kernel(x_1, x_2)

$f_i = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$

return

⇒ Make sure to perform feature scaling before using Kernel.

> Any kernel choice must satisfy "Mercer's Theorem" to make sure the SVM's optimization works correctly.

> Other Kernels: 1) Polynomial kernel: $K(x, l) = (x^T l)^2, (x^T l)^3, (x^T l + 5)^4, \ldots$

$= (x^T l + \text{const})^{\text{degree}}$

2) String Kernel

3) Chi-square kernel.

4) histogram intersection kernel

Multiclass Classification

- Packages come with built in multiclass classification.
- Otherwise use One vs. All method.
  Take K SVMs, one to distinguish $y=i$ from the rest...

Logistic Regression VS. SVMs

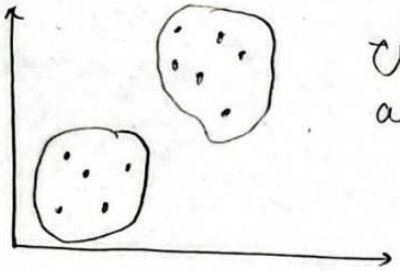
- If n is large (relative to m):
  → Use logistic regression, or SVM without a Kernel.
- If n is small (1-1000) & m is intermediate (10-10,000).
  → Use SVM with Gaussian.
- If n is small, m is large: (→ (100,000s))
  → Create/add more features, then use logistic regression or SVM without a Kernel.
- NN if well designed can work in all cases, but takes time to train

→ SVM optimization is convex so we don't care about local optimum
Unlike in NN.. Even in NN local optima are rarely a problem

Turn upside down

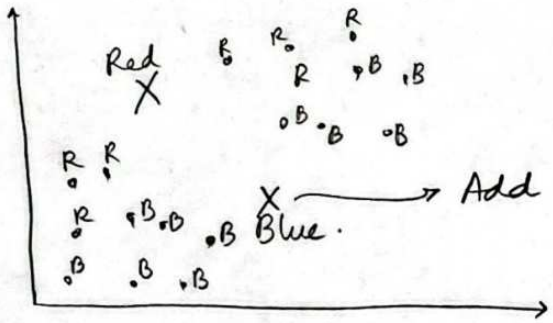
Unsupervised learning.

Lec 13.1 : Clustering.



Clustering algorithm.

Lec 13.2 K-means algorithm.



Step ① :- Assign cluster
Assign each pt to Red or Blue depending on which is closer

Step ② :- Move centroid
Move them to the avg. of all pts having same colour.

→ Repeat these steps. until convergence ⇒ no change in colour of centroid.

K-means Algorithm.

Input

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

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

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

Repeat {

  for $i = 1$ to m — cluster assignment

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

  for $K = 1$ to K — centroid update

    $\mu_K :=$ average of pts assigned to cluster K.

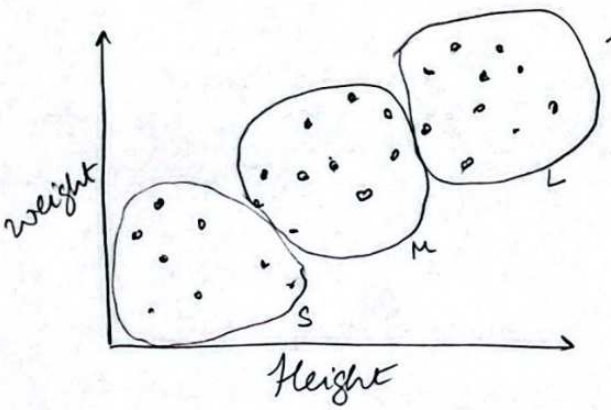
}

> $c^{(i)} = \min_K \| x^{(i)} - \mu_K \|^2$

Value of K that minimizes distance to clusters.

> Eliminate a cluster centroid if nothing gets assigned to it or randomly reassign clusters.

> <u>K means for non seperated clusters.</u>



→ Cluster T shirts into S, M, L.

> K means does this.

> Market segmentation.

Lec 13.3 : Optimization Objective of K means.

- $c^{(i)}$ = index of cluster $(1, 2 \ldots K)$ to which example $x^{(i)}$ is assigned
- μ_K = cluster centroid K. $(\mu_K \in \mathbb{R}^n)$
- $\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned.

Optimization Objective.

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

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

> J is called distortion/cost fn. We find c, μ to minimize J.
> This is the same as cluster assignment & move centroid.

> Cluster assignment $\equiv$ min J w.r.t $c^{(1)}, c^{(2)} \ldots c^{(m)}$
> Move centroid $\equiv$ min J w.r.t $\mu_1, \mu_2 \ldots \mu_K$.

} Can be shown mathematically

Lec 13.4 Random Initialization.

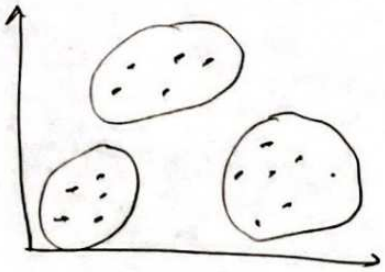
→ How to initialize cluster centroids?

> Should have $K < m$ (of course)
> Randomly pick K training examples.
> Set $\mu_1, \ldots, \mu_K$ equal to these K examples.

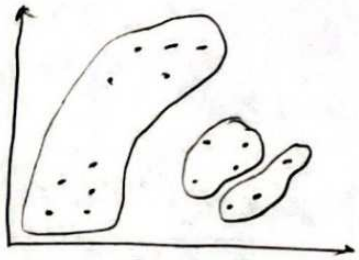
} Basically initialize centroids on 2 training examples.

⇒ $\mu_1 = x^{(i)}$
$\mu_2 = x^{(j)}$ & so on...

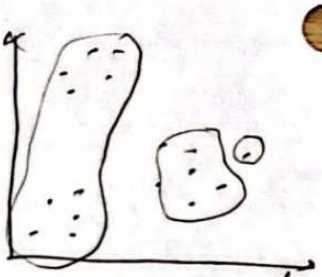
> Depending on initialization, K means can converge to different clusters => different local optima of J.



Global Optimum



one local optimum



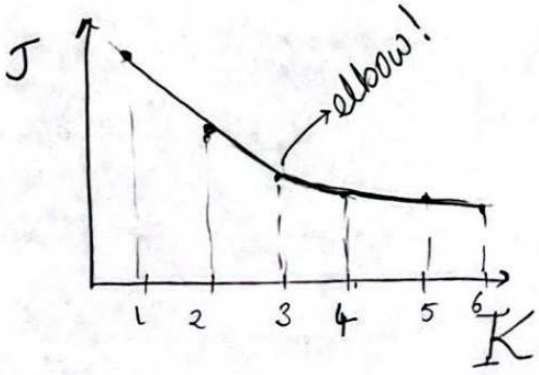
another local optimum.

> Try multiple random initializations & look for best solution
> 50 - 1000 times! Pick the one with lowest distortion J.
> Works better for smaller no. of clusters $K \cong 2-10$.

Lec 13.5 : Choosing the number of clusters K.

> Best way is visual inspection

> Elbow method



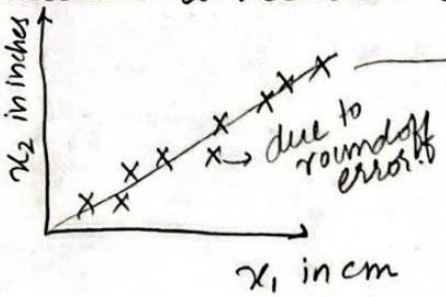
> More often there is no clear elbow.
⇒ Worth a shot but not robust.

> K could be chosen based on application.

Lec 14.1 Dimensionality Reduction. → Unsupervise learning.

(I) Data compression

- Reduce correlated data from 2D to 1D.



> Project all points onto the line & use the location on line as a new feature.

Lec 14.2 (II) Data visualization.

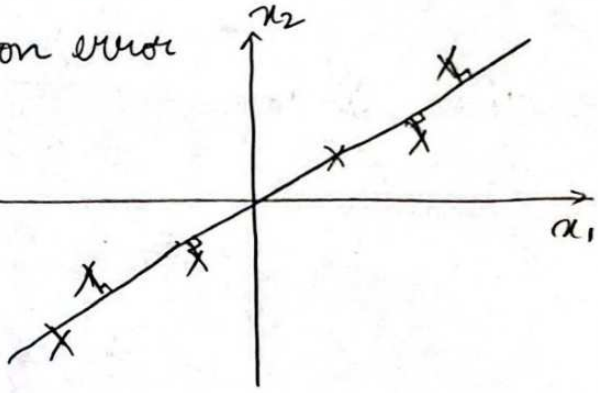
> How can we plot data with large number of features.

> Reduce 50 features to 2 features, where each feature represents some essence of the original ones.

Lec 14.3 Principle Component Analysis (PCA)

→ Project 2D onto line S.T the projection error is minimum.

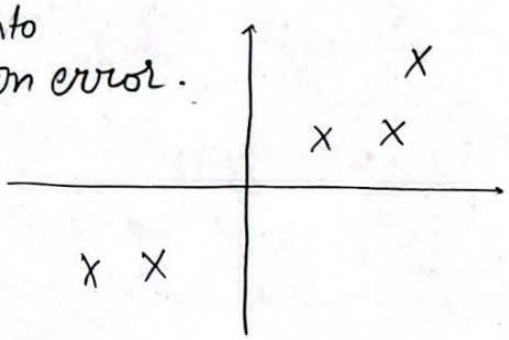
> First perform mean normalization and feature scaling.



Problem Formulation.

> Find vector $u^{(1)}$ S.T data projected onto line through $u^{(1)}$ gives least projection error.

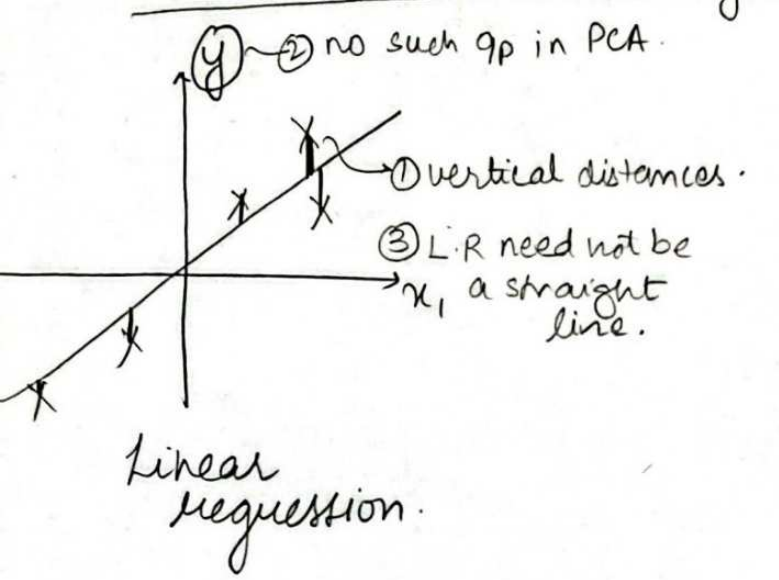
> Could use $u^{(1)}$ or $-u^{(1)}$.



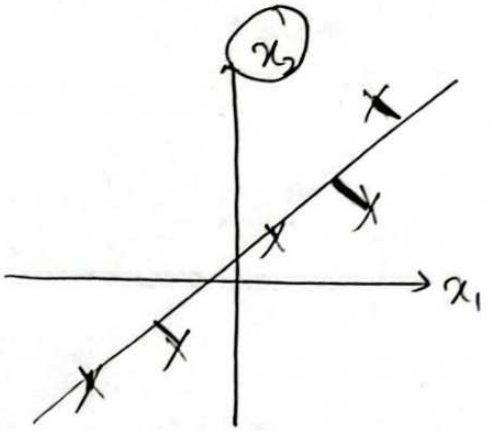
→ To reduce n-dimensions to K-dimensions. Find K vectors $u^{(1)}, u^{(2)} \ldots u^{(K)}$ onto which to project the data.

→ Project onto the span of vectors $u^{(1)}, u^{(2)} \ldots u^{(K)}$.

→ <u>PCA is not linear regression.</u>



Linear regression.



PCA.

Lec 14.4 PCA Algorithm

Data preprocessing.

- Perform feature scaling and <u>mean normalization.</u> more important.

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

Replace $x_j^{(i)}$ with $x_j - \mu_j$

- Want to go from $x^{(i)} \in \mathbb{R}^n$ to $z^{(i)} \in \mathbb{R}^k$
- Need to find $u^{(i)} \rightarrow$ vectors that form the span to project onto.
  - also the projections $z^{(i)}$

Procedure / Algorithm.

- Compute "covariance matrix"

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

(Σ: $n \times n$ matrix; $x^{(i)}$: $n \times 1$; $(x^{(i)})^T$: $1 \times n$)

Vectorized form:

$$\text{Sigma} = (1/m) \cdot X' \cdot X$$

↳ Works if X = design matrix = $\begin{bmatrix} — x^{(1)T} — \\ — x^{(2)T} — \\ — x^{(m)T} — \end{bmatrix}$

- Compute eigenvectors of matrix Σ:

```
» [U,S,V] = svd (Sigma);
```

- This works because the covariance matrix is symmetric positive semidefinite.
- Columns of U are the eigenvectors $u^{(1)}, u^{(2)} \ldots . u^{(n)}$
- Take the first K of these $u^{(n)}$ vectors.

- We have $\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(K)} \\ | & | & & | \end{bmatrix}$ → Ureduce

$n \times K$

- $$z = U_{reduce}^T \cdot x$$

($K \times 1$) ($K \times n$) ($n \times 1$)

(Math proof is not provided)

Algorithm

```
» sigma = (1/m) * X' * X;
» [U,S,V] = svd (sigma);
» Ureduce = U(:, 1:K);
» z = Ureduce' * x;
```

Lec 14.5 Choosing the no. of principle components.

Average squared projection error : $\frac{1}{m}\sum_{j=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2$

Total variation in the data : $\frac{1}{m}\sum_{j=1}^{m} \| x^{(i)} \|^2$

Typically, choose K to be smallest value so that,

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approx}^{(i)} \|^2}{\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} \|^2} \leq 0.01 \qquad 1\%.$$

$\Rightarrow$ "99% of variance is retained"

Algorithm

- Try PCA with $K = 1$
- Compute U_{reduce} & z & x_{approx} (explained in 14.6)
- Check if $\frac{\frac{1}{m}\Sigma \cdots}{\frac{1}{m}\Sigma \cdots} \leq 0.01$
- Increment K & Keep checking.

- A better approach is to use the S matrix from SVD.

$$S = \begin{bmatrix} S_{11} & & & 0 \\ & S_{22} & & \\ & & S_{33} & \\ 0 & & & \ddots \\ & & & & S_{nn} \end{bmatrix}$$

↳ Eigenmatrix!

→ SVD needs to be called only once.

Not proven here but the value

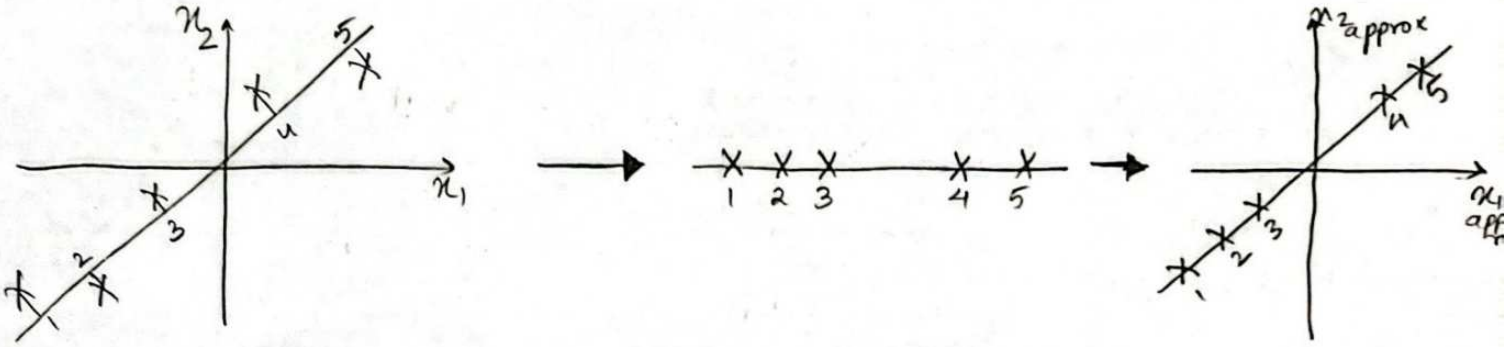
$$\frac{\frac{1}{m}\Sigma \cdots}{\frac{1}{m}\Sigma \cdots} = 1 - \frac{\sum_{j=1}^{K} S_{jj}}{\sum_{j=1}^{n} S_{jj}}$$

$\Rightarrow$ Just test $\boxed{\frac{\sum_{i=1}^{K} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99}$

Lec 14.6 Reconstruction from Compressed Data.

- Recall $z = U_{reduce}^T x$. $\quad z \in \mathbb{R}^k$, $x \in \mathbb{R}^n$

- $\boxed{X_{approx} = U_{reduce} \cdot z}$ $\quad$ U is Symmetric true semi definite.



Lec 14.7 Advice for applying PCA.

Supervised learning speedup

- Extract inputs: Unlabeled dataset $x^{(1)}, x^{(2)} \ldots x^{(m)} \in \mathbb{R}^n$

$\downarrow$ PCA

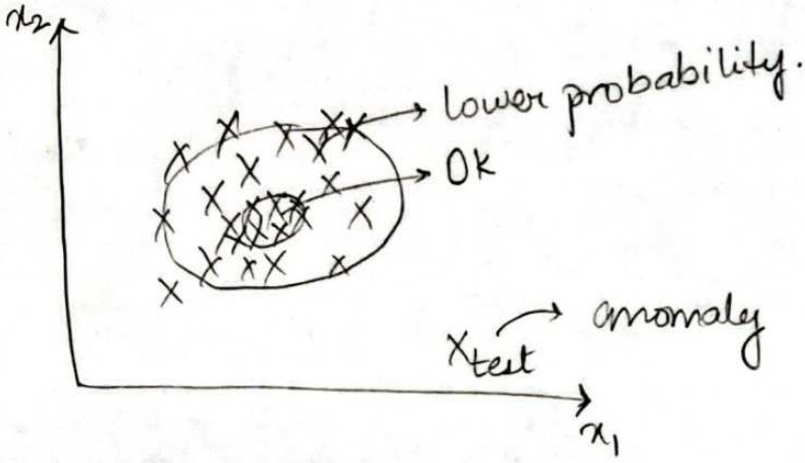
$z^{(1)}, z^{(2)} \ldots z^{(m)} \in \mathbb{R}^k$

- Use z to train a NN or use any learning algorithm.

- The mapping from $x \rightarrow z$ should be defined only using the training set. CV & test set can use this mapping after it has been defined.

- Do NOT use PCA to prevent overfitting just because it reduces the no. of features! Use regularization instead.

Lec 15.1 - Anomaly Detection.



> Given dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

> Is x_{test} anomalous?

> Build a model $p(x) \longrightarrow$ probability of x.

> If $\begin{cases} p(x_{test}) < \epsilon \longrightarrow \text{flag anomaly.} \\ p(x_{test}) \geq \epsilon \longrightarrow \text{OK.} \end{cases}$

> Uses: Fraud detection, manufacturing, monitoring computers in a data center.

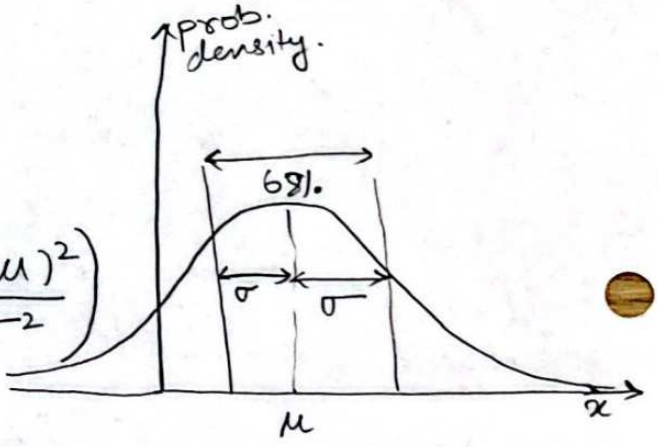
Lec 15.2 Gaussian Distribution (aka Normal)

Say $x \in \mathbb{R}$., mean μ & variance σ^2.

$x \sim \mathcal{N}(\mu, \sigma^2)$ ← normal dist

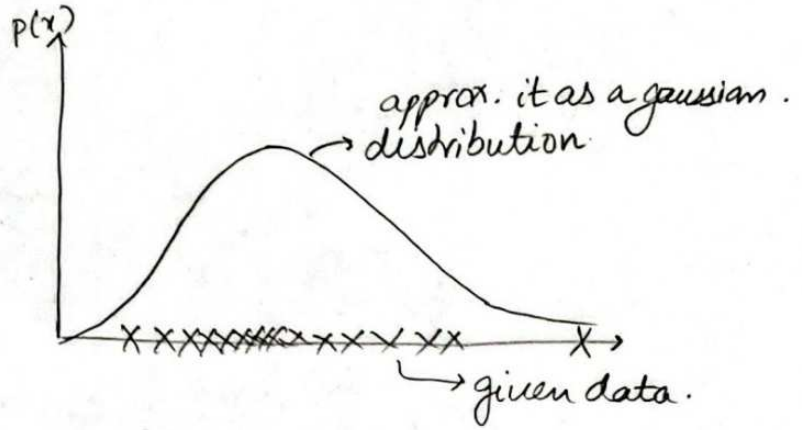
↑ "distributed as"

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\,\sigma} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



→ Area under curve should be 1. since $\int_{-\infty}^{\infty} p(x) = 1$

- Given dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $\quad x^{(i)} \in \mathbb{R}$.



$x^{(i)} \sim N(\mu, \sigma^2)$

$$\mu = \frac{1}{m} \sum_{j=1}^{m} x^{(i)}$$

- $$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

Lec 15.3 - Anomaly Detection Algorithm.

Training set : $\{x^{(1)}, \ldots, x^{(m)}\}$.
Each example is $x \in \mathbb{R}^n$.

$$p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2)\, p(x_3; \mu_3, \sigma_3^2) \ldots p(x_n; \mu_n, \sigma_n^2).$$

→ Here we are assuming each feature has a Gaussian distribution. Also we assume all features are independant. The algorithm still works even if they are not independant.

- $$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

Anomaly detection Algorithm.

1. Choose features x_j that you think might be indicative of anomalous examples.

2. Fit parameters $\mu_1, \mu_2 \dots, \mu_n; \sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

3. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(\frac{-(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \epsilon$.

Lec 15.4 Developing and Evaluating An Anomaly Det. System.

- Need real number evaluation.
- Assume we have some labeled data, of anomalous & non anomalous examples ($y=0$ if normal, $y=1$ if anomalous)
- Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ (assume normal examples/ not anomalous)
- Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

  Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

> Say we have: 10,000 normal data points.
20 anomalous " "

> Training set: 6000 good points.
CV set: 2000 good points, 10 anomalous ($y=1$)
Test set: 2000 good points, 10 anomalous ($y=1$)

Algorithm evaluation.

> Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$

> On a cross validation/test example x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \epsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \epsilon \text{ (normal)} \end{cases}$$

> Possible evaluation metrics:
- True positive, false positive, false neg, true neg.
- Precision/Recall
- F1 score.

> Can also use cross validation set to choose parameter ϵ.

Lec 15.5 Anomaly detection vs. Supervised learning

Anomaly detection

- Small no. of positive examples ($y = 1$) $\Rightarrow$ anomaly.
- Large no. of negative examples ($y = 0$).
- Many different "types" of anomalies.
- Fraud detection for small no. of fraud cases.

Supervised learning

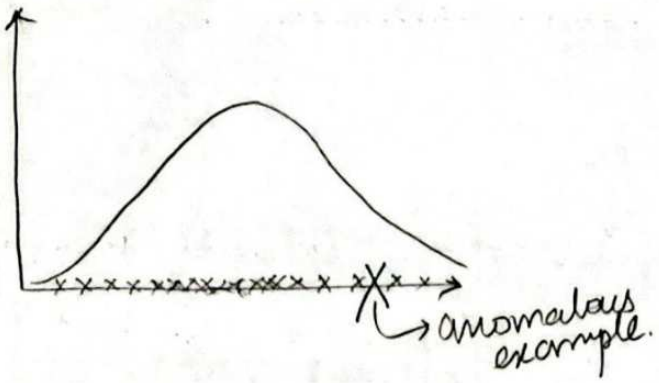
- Large number of positive and negative examples.
- If future positive examples look "similar" to earlier ones
- Email spam classification.
- Weather prediction, cancer classification.

Lec 15.6 Choosing what features to use.

- If the dataset doesn't look gaussian, the algorithm may still work but it is better to use some transformation on the data and make it "look" gaussian. ($\log(x)$ for example)
- In general we could try $\log(x + c)$ (c: constant)
  - $\sqrt{x}$, $x^{1/3}$, ...
- Error analysis to come up with features.

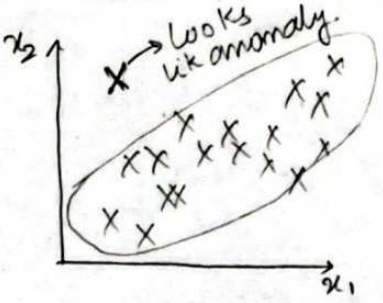
Want $p(x)$ large for normal examples x.
$p(x)$ small for anomalous examples x.

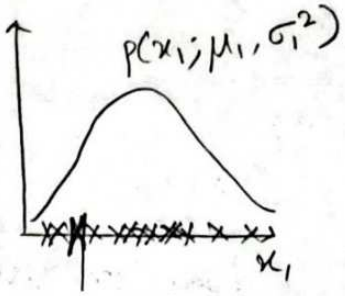
Most common problem:
$p(x)$ is comparable (say, both large) for normal and anomalous examples.

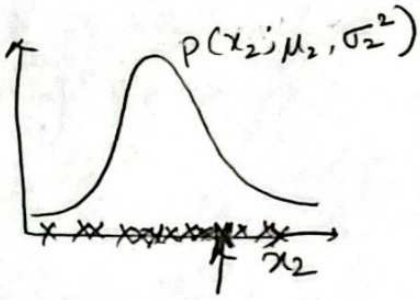


> In this case look for a feature that seperates the anomaly from the normal values.

Lec 15.7 Multivariate Gaussian Distribution.







→ The anomaly lies within the Gaussian of both x_1 & x_2.

→ Current algorith uses each feature seperately and fails to identify the anomaly. Algorithm uses circles for locus of equal probability.

→ Multivariate gaussian distribution.

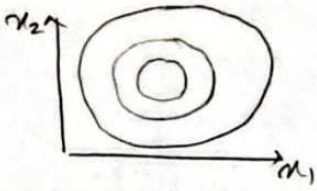
> $x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2)$... etc seperately. Model $p(x)$ all in one go.

> Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix).

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$|\Sigma|$ → determinant of Σ.

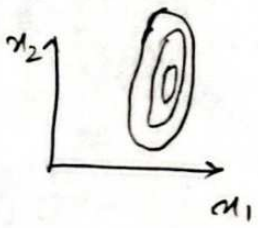
Eg:- $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\Rightarrow$



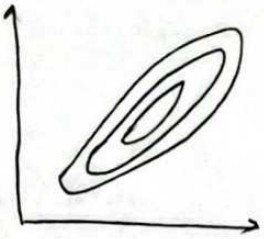
> Symmetric bell in x_1 & x_2.

> $\Sigma \downarrow \Rightarrow$ taller & narrower symmetric bell.

> $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$ $\Rightarrow$ x_1 variance is reduced.

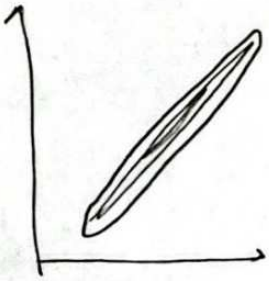


> $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ $\Rightarrow$

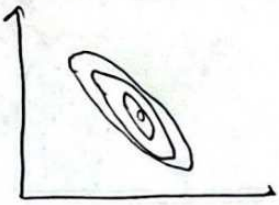


$\Rightarrow$ x & y grow together.
> Models the correlation.

> $\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$

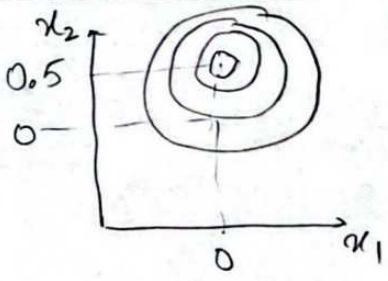


> $\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$



$\Rightarrow$ -ve correlation.

> $\mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 0 \\ 0 & 1 \end{bmatrix}$



> Moves the peak to (0, 0.5)

Lec 15.8 Anomaly detection using Multivariate Gaussian Distribution.

$$p(x;\mu,\Sigma)=\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right).$$

Estimating μ & Σ

$$\mu=\frac{1}{m}\sum_{j=1}^{m}x^{(i)}\;;\quad \Sigma=\frac{1}{m}\sum_{j=1}^{m}(x^{(i)}-\mu)(x^{(i)}-\mu)^T$$

↳ same as PCA.

Algorithm

1. Fit model $p(x)$ by setting

$$\mu=\frac{1}{m}\sum_{i=1}^{m}x^{(i)}$$

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

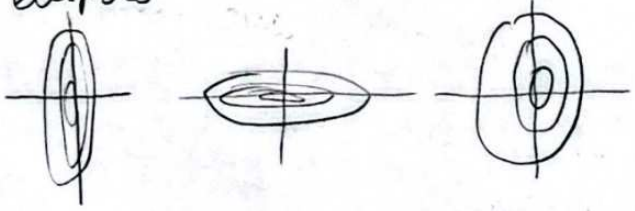
2) Given a new example x, compute

$$p(x)=\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Flag an anomaly if $p(x)<\epsilon$.

Original model: $p(x)=p(x_1;\mu_1,\sigma_1^2)\times p(x_2;\mu_2,\sigma_2^2)\times\dots\times p(x_n;\mu_n,\sigma_n^2)$

> Mathematically it can be proven that this model corresponds to multivariate gaussian distributions that are axis aligned.
⇒ ellipses that are not tilted.



⇒ Same as having Σ with 0 in off diagonal elements & Σ has σ^2 on diagonal.

$$\Sigma=\begin{bmatrix}\sigma_1^2 & & 0\\ & \sigma_2^2 & \\ 0 & & \ddots\end{bmatrix}$$

> Use multivariate when features have a correlation, but it is computationally expensive due to Σ^{-1}, it also needs large m. (m must be $\gg n$).

Lec 16.1 Recommender Systems.

> Rate movies from 0 to 5 stars.

Movie		Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
#1	romance	5	5	0	0	0.9	0
#2	romance	5	?	?	0	1.0	0.01
#3	romance	?	4	0	?	0.99	0
#4	Action	0	0	5	4	0.1	1.0
#5	Action	0	0	5	?	0	0.9

n_u = no. of users = 4

n_m = no of movies = 5

$r(i,j)$ = 1 if user j has rated movie i

$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j)=1$)

> The recommender system fills in the ? data & based on that recommendations are made.

> Features x_1 & x_2 denote the degree to which a movie is romance or action respectively.

> For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^{n+1}_{3}$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars $\Rightarrow$ Regression Problem with x_1, x_2 as features & existing ratings as output y.

> What is Alice's rating for movie #3?

$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$ from x_1, x_2. Say, $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$

$\Rightarrow (\theta^{(1)})^T x^{(3)} = 4.95$

Problem Formulation

$r(i,j) = 1$ if user j has rated movie i (0 otherwise)

$y^{(i,j)}$ = rating by user j on movie i (if defined).

$\theta^{(j)}$ = parameter vector for user j

$x^{(i)}$ = feature vector for movie i

For user j, movie i, predict rating: $(\theta^{(j)})^T (x^{(i)})$.

$m^{(j)}$ = no. of movies rated by user j.

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)}\right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

$\theta^{(j)} \in \mathbb{R}^{n+1}$.

> We ignore $m^{(j)}$.

> Summarizing for all users $\theta^{(1)}, \theta^{(2)} \dots \theta^{(n_u)}$:

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \underbrace{\frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2}_{J(\theta^{(1)}, \dots, \theta^{(n_u)})}$$

Gradient descent update

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} \quad (\text{for } k=0)$$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \text{ for } k \neq 0$$

> This algorithm is called content based recommender system since we need to define x ourself based on the content.

Lec 16.3 Collaborative Filtering.

> The algorithm learns the features!

> Say we ask the users to rate how much they like romance or action.

Alice: $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ (5 → romance!); Bob: $\theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$ (0 for $x_0=1$); $\theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$ (5 → action!); $\theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$

→ Idea: Romantic movie lovers (Alice, Bob) would highly rate romantic movies ⇒ define features accordingly.

⇒ What should $x^{(1)}$ be s.t $(\theta^{(1)})^T x^{(1)} \cong 5$

$(\theta^{(2)})^T x^{(1)} \cong 5$

$(\theta^{(3)})^T x^{(1)} \cong 0$

$(\theta^{(4)})^T x^{(1)} \cong 0$

Optimization algorithm.

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)} \ldots x^{(n_m)}$:

$$\min_{x^{(1)}, \ldots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

> Regression: Given $x \longrightarrow$ predict θ.

> Collaborative filtering: Given $\theta \longrightarrow$ predict x.

> What we can do is guess $\theta \xrightarrow[\text{filt}]{\text{cd}}$ predict x.

predict $\uparrow$ — lin reg. $\leftarrow$

> Running this loop over time actually gives good predictions of both θ & x.

Lec 16.4 The real Collaborative filtering algorithm.

> A more efficient way to solve for θ & x simultaneously.

> Minimize a new cost fn to get both θ & x.

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

$$\min_{\substack{x^{(1)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \dots, \theta^{(n_u)}}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

→ In this case there is no $x_0 = 1$ ⇒ $x \in \mathbb{R}^n$ & $\theta \in \mathbb{R}^n$.

Collaborative filtering algorithm.

1. Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values.

2. Minimize $J(x^{(1)}, \dots \theta^{(n_u)})$ using gradient descent (or other).

   Eg: for every $j = 1 \dots n_u \quad i = 1 \dots n_m$:

$$x_k^{(i)} = x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} = \theta_k^{(j)} - \alpha \left(\sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$. $\Rightarrow (\theta^{(j)})^T (x^{(i)})$

Lec 16.5 Vectorization, low Rank Matrix Factorization.

> Group rating data in a matrix.

$$Y = \begin{matrix} \\ n_m \downarrow \end{matrix} \overset{n_u \rightarrow}{\begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}}$$

Predicted rating:

$$X\Theta = \begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \cdots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & & & \\ \vdots & & & \\ (\theta^{(1)})^T(x^{(n_m)}) & \cdots & \cdots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

Where,

→ $$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(n_m)})^T- \end{bmatrix} \qquad \Theta = \begin{bmatrix} | & | & & | \\ \theta^{(1)} & \theta^{(2)} & \cdots & \theta^{(n_u)} \\ | & | & & | \end{bmatrix}$$

→ This algorithm is called low rank matrix factorization. Since $X\Theta$ is low rank.

> For each product (movie) i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$

Say x_1 = romance, x_2 = action, x_3 = comedy etc...

> How to find movies j related to movie i?

> $\|x^{(i)} - x^{(j)}\|$ is small $\Rightarrow$ $x^{(i)}$ & $x^{(j)}$ are similar movies

Lec 16.6 Implementation detail: Mean normalization.

> If a new use Eve has not rated any movie ...

$$Y = \begin{bmatrix} \theta^{(1)} & \theta^{(2)} & \theta^{(3)} & \theta^{(4)} & \theta^{(5)} \\ 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

Say $n=2$.

$\Rightarrow$ First time has no role and $\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ since regularization term can freely min θ. $\Rightarrow$ Eve will rate every movie as 0. Which is wrong.

→ Use mean normalization.

Calculate mean rating of each movie ...

$$\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$. Subtract μ from Y

$$\Rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

→ Use this Y! Make sure to add back the mean when making the predictio

⇒ Prediction: $(\theta^{(j)})^T (x^{(i)}) + \mu_i$

⇒ User 5 (Eve):

$$\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow (\theta^{(5)})^T (x^{(i)}) + \mu_i$$

gives prediction $\begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$

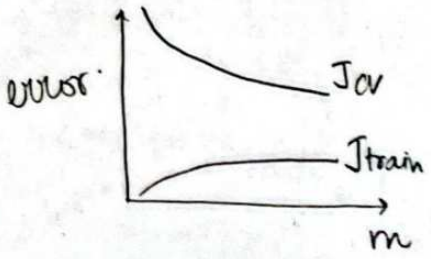
⇒ Prediction is actually μ.

> Movie with no ratings ⇒ could take avg. of columns. ⇒ But is it worth recommending it at all?

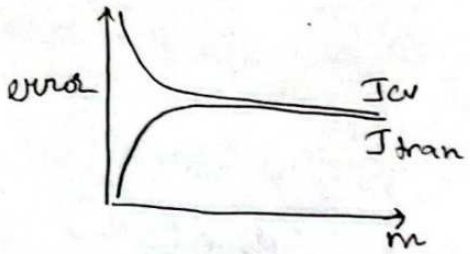
Lec 17.1 Large Scale Machine learning.

> Improve gradient descent to deal with large m ≈ billion.

> Check learning curves to see if large m is required.



⇒ High variance
↑↑ m is good.



⇒ High bias
Need more features.

Lec 17.2 Stochastic Gradient descent.

Linear regression with gradient descent

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j \qquad J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

} ↳ Called Bath grad. descent since we look at all m.

Stochastic Gradient descent

→ $\text{cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$

$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}(\theta, (x^{(i)}, y^{(i)}))$

1. Randomly shuffle dataset
2. Repeat { → This outer loop runs very few times (1-10 times)

for $i = 1, \ldots, m$ { → $\frac{\partial}{\partial \theta_j} \text{cost}(\theta, (x^{(i)}, y^{(i)}))$

$\theta_j = \theta_j - \alpha \underbrace{(h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}}$

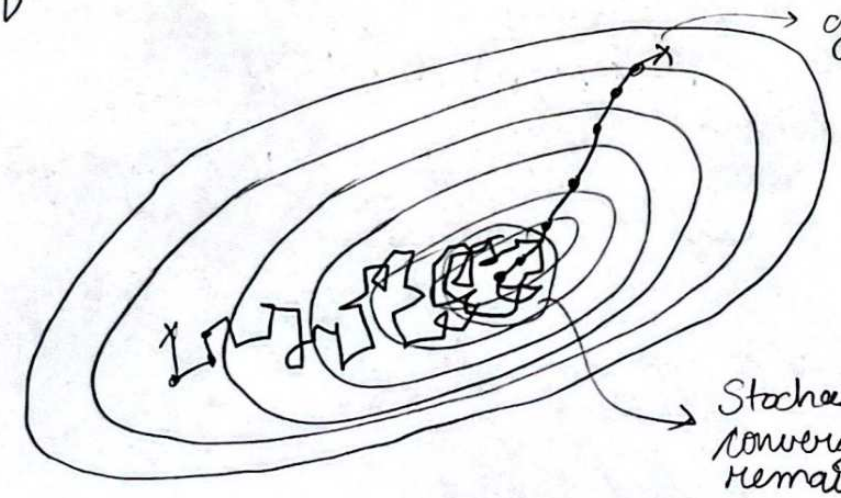
(for $j = 0, \ldots, n$)

}

}

→ This looks at each data point & makes adjustments to the parameters at each point one at a time.

→ Progress to improve the parameters doesn't have to wait for the summation term to be computed.



Lec 17.3 Mini batch gradient descent.

> In between stochastic & batch ⇒ use b examples in each iteration.

b = mini batch size. ($b \approx 10$; usually $\approx 2-100$)

Get $b = 10$ examples. $(x^{(i)}, y^{(i)}) \ldots (x^{(i+9)}, y^{(i+9)})$.

Algorithm assume $b = 10$, $m = 1000$.

Repeat {

for $i = 1, 11, 21, 31 \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} \left(h_\theta(x^{(k)}) - y^{(k)}\right) x_j^{(k)}$$

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

}

}

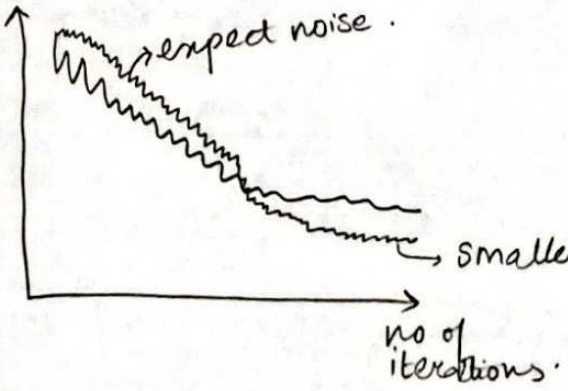
> Could be faster than both batch & stochastic if mini batch can be implemented in a vectorized way properly.

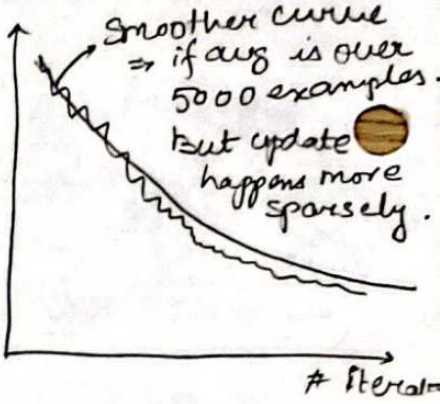
Lec 17.4 Stochastic gradient descent convergence.

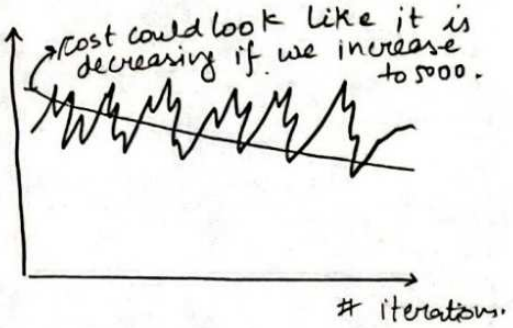
> During learning, compute cost $(\theta, (x^{(i)}, y^{(i)}))$, before updating θ using $(x^{(i)}, y^{(i)})$

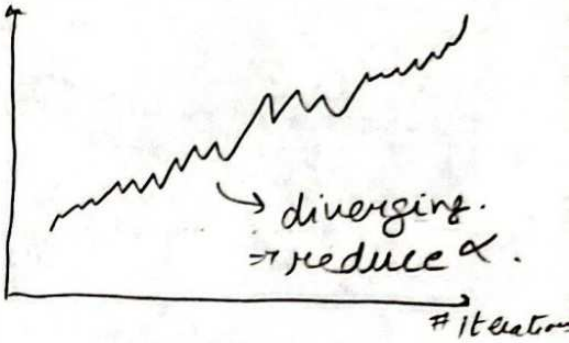
> Every 1000 iterations (say), plot cost $(\theta, x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by the

Checking for convergence.









> Could lower α over time to end up in convergence.

$$\alpha = \frac{\text{const 1}}{\text{iteration no.} + \text{const 2}}$$

Lec 17.5 Online Learning.

> Dealing with a continuous stream of data.

> Customers come to website for shipping based on price they decide to ship ($y=1$) or not to ship ($y=0$).

> We want to learn $p(y=1 \mid x;\theta)$ to optimize price. Use logistic regression for $p(y=1)$.

Algorithm

> Repeat forever {

Get (x, y) corresponding to user.

Update θ using (x, y):

$\theta_j := \theta_j - \alpha (h_\theta(x) - y) x_j \quad (j = 0, \ldots, n)$.

}

> Shows another simple algorithm using Click Through Rate (CTR).

Lec 17.6 Map Reduce and Data Parallelism.

Say Batch gradient descent ($m = 400$)

$$\theta_j = \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Split training set

Machine 1: Use $(x^{(1)}, y^{(1)}) \ldots (x^{(100)}, y^{(100)})$

& compute, $temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$

Machine 2: Use $(x^{(101)}, y^{(101)}) \ldots (x^{(200)}, y^{(200)})$

& compute $temp_j^{(2)}$

⋮

Machine 4: Compute $temp_j^{(4)}$.

> Take the 4 temp variables & combine in another server.

$$\theta_j = \theta_j - \alpha \frac{1}{400} \left(temp_j^{(1)} + temp_j^{(2)} + \dots + temp_j^{(4)} \right)$$

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

> Shows how map reduce can be used for advanced optimization algorithms.

> Could use multiple cores inside a single computer.

Lec 18 : Applies ML to an example : Photo OCR.

> He also briefly talks about ceiling analysis in pipelined machine learning applications.