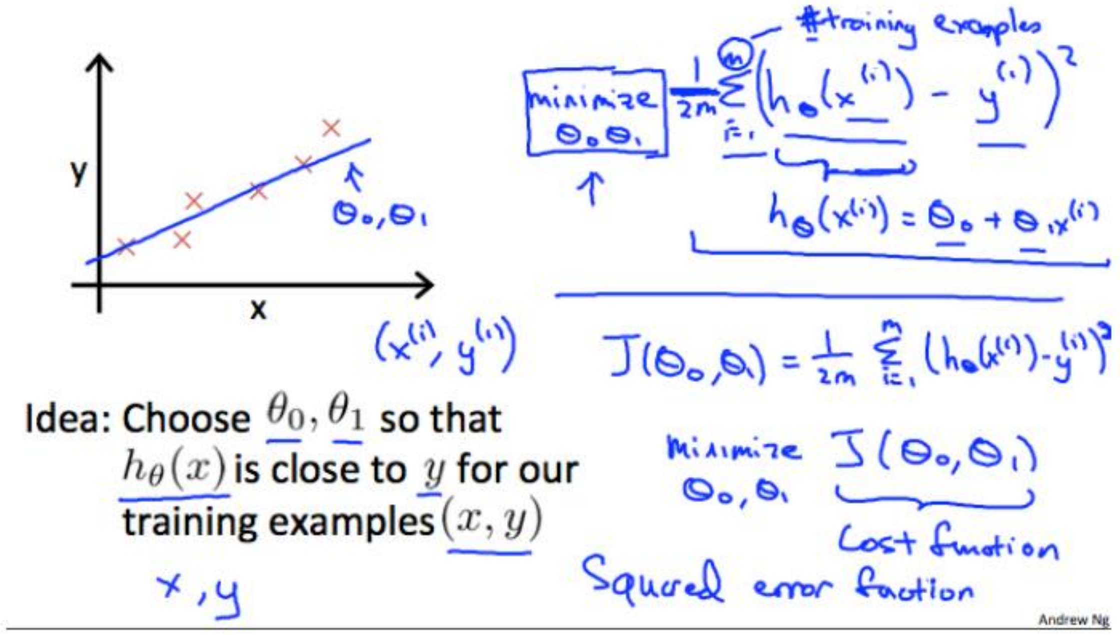
Week 1:

1. Algo and math not just enough, application is imp
2. ML grew out of AI
3. Examples:
   a. Click stream data used know more about users
   b. Medical record to medical knowledge
   c. Genomics
   d. Autonomous helicopter (can't be programmed)
   e. NLP
   f. Handwriting recognition
   g. CV
   h. Recommendation systems
4. ML – ability to learn without being explicitly programmed (Checkers example)
5. Supervised Learning – labelled dataset (for every data point we had "right" answer)
   a. Classification / Regression
6. Unsupervised Learning – don't give "right" answers
   a. Clustering
      i. Google news (cohesive news clustering)
      ii. Clustering people into group from gene data
      iii. Social network analysis
      iv. Market segmentation
   b. Cocktail Party problem – hard to hear people in noisy party
      i. Multiple speaker recording multiple people's audio
      ii. Pass them to algo, then algo will split the audios
7. Model Representation – Regression Supervised problem (hypothesis)
8. Cost Function – Help fit best line to data (choosing parameters of model changes hypothesis function)
   a. Linear regression – we need to determine 2 parameters θ_0 and θ_1 so as to minimize squared error of all the training examples

$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$



Idea: Choose $\underline{\theta_0}, \underline{\theta_1}$ so that $\underline{h_\theta(x)}$ is close to $\underline{y}$ for our training examples $\underline{(x, y)}$

x, y

$$\boxed{\text{minimize}\ \theta_0\theta_1}\ \frac{1}{2\overset{\#\text{training examples}}{m}}\sum_{i=1}^{m}\left(\underbrace{h_\theta(x^{(i)})}_{} - \underline{y^{(i)}}\right)^2$$

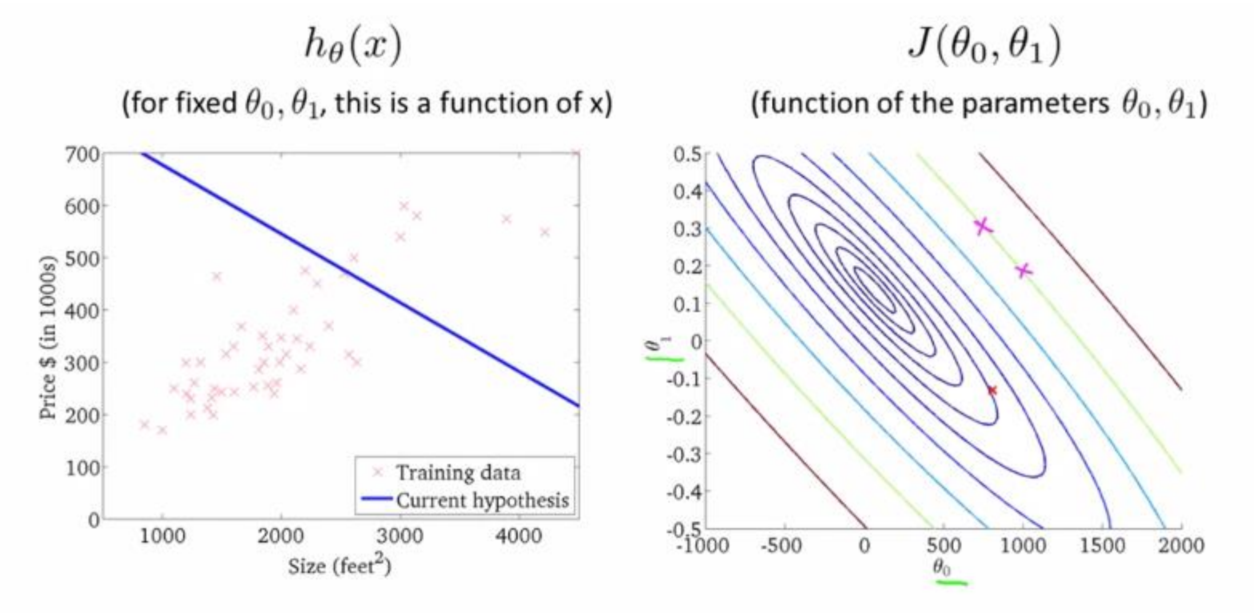
$$\underbrace{h_\theta(x^{(i)}) = \underline{\theta_0} + \underline{\theta_1} x^{(i)}}_{}$$

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

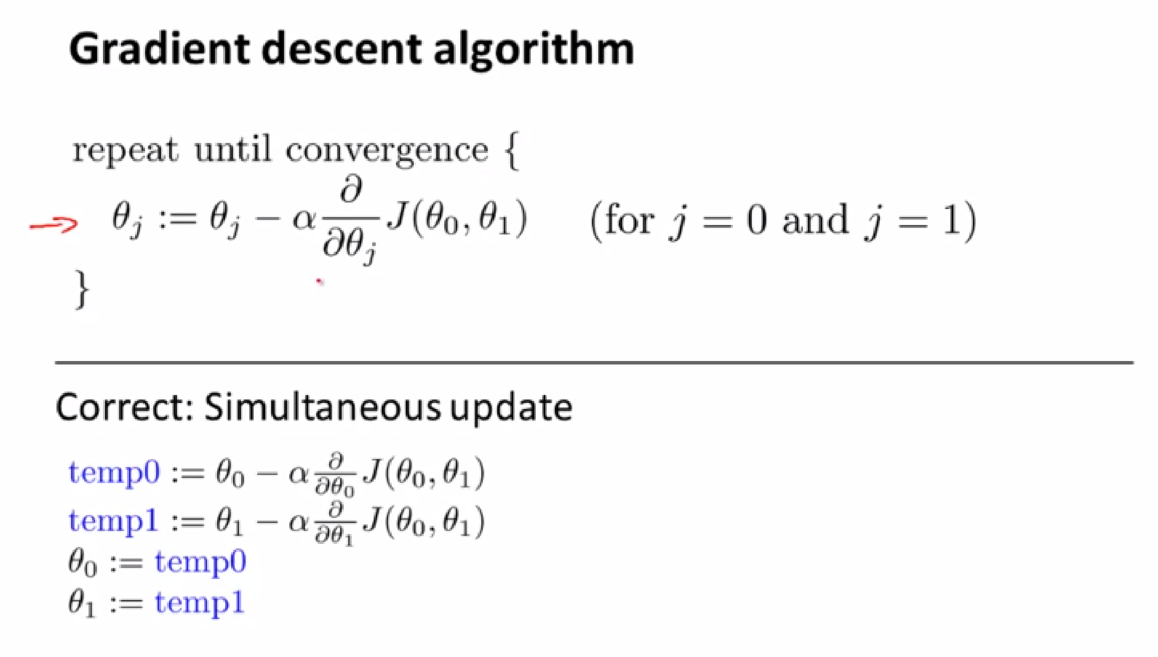
$$\underset{\theta_0,\theta_1}{\text{minimize}}\ \underbrace{J(\theta_0,\theta_1)}_{\text{Cost function}}$$

Squared error function

   b. Contour plots used for better visualizing cost function minima point



9. Gradient Descent – used to minimize cost function (in linear regression). It applied to any general cost function
    a. Vary θ_0 and θ_1 and go towards local minima



Gradient descent algorithm

repeat until convergence {

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

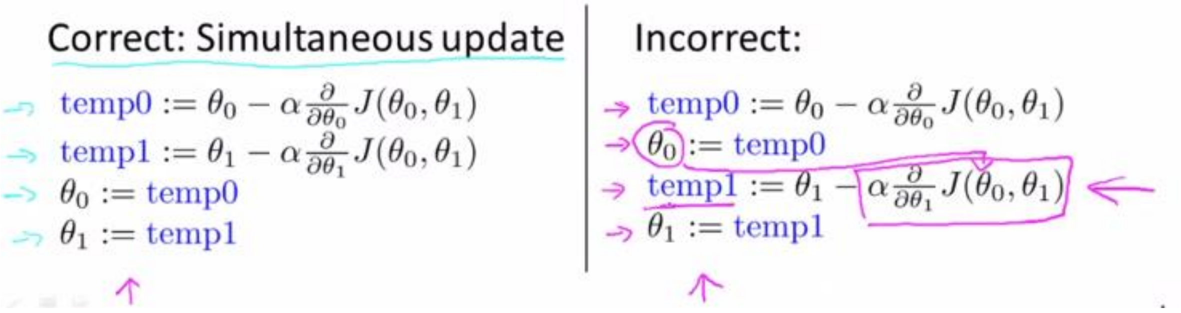
}

Correct: Simultaneous update

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

Alpha – learning rate, derivative of cost function (slope of tangent)

    b. Correct way to implement gradient descent (simultaneous update)



Correct: Simultaneous update

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

Incorrect:

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

    c. For linear regression, we will always have a convex function for cost

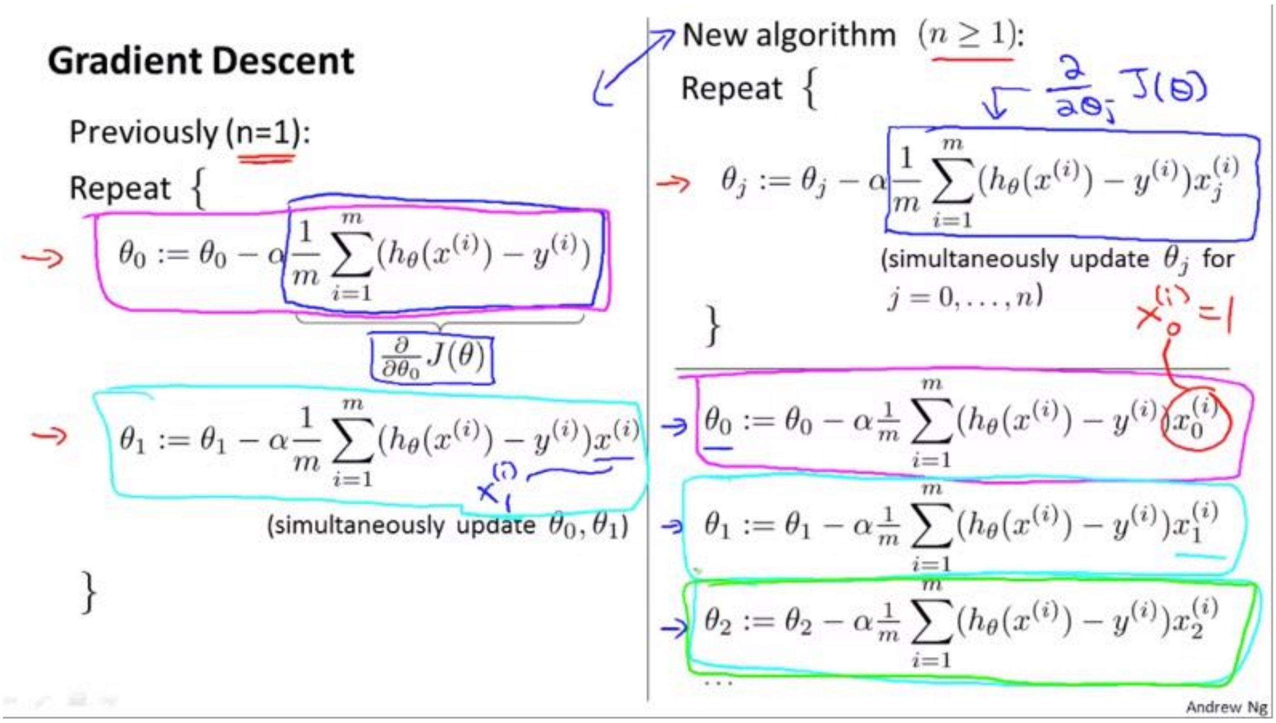
10. Linear Algebra:
    a. Matrix: dimensions
    b. Vector: n x 1 matrix, 1 indexed (1,2, 3….), 0 indexed (0,1, 2…)
    c. Matrix Algebra: addition, scalar multiplication, vector multiplication

    i. Express linear regression hypothesis function as vector multiplication
    ii. Prediction = data matrix * parameters
    iii. Method to apply multiple hypothesis on dataset (linear regression)

d. Matrix properties:
    i. A X B not equal to B X A
    ii. A X (B X C) = (A X B) X C
    iii. A X I = I X A = A
    iv. Matrix which don’t have inverse are singular/ degenerate

Week 2:

1. Multiple Features (Multivariate linear regression):
   a. n = number of features, h depends on $\theta_0 \theta_1 \theta_2 \theta_3$
   b. To accommodate the constant term in the equation of hypothesis, theta matrix is represented in 0 indexed notation consisting of n+1 term
2. Gradient Descent for multiple variables:



Gradient Descent

Previously (n=1):

Repeat {

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

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

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

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

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

}

$x_0^{(i)} = 1$

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

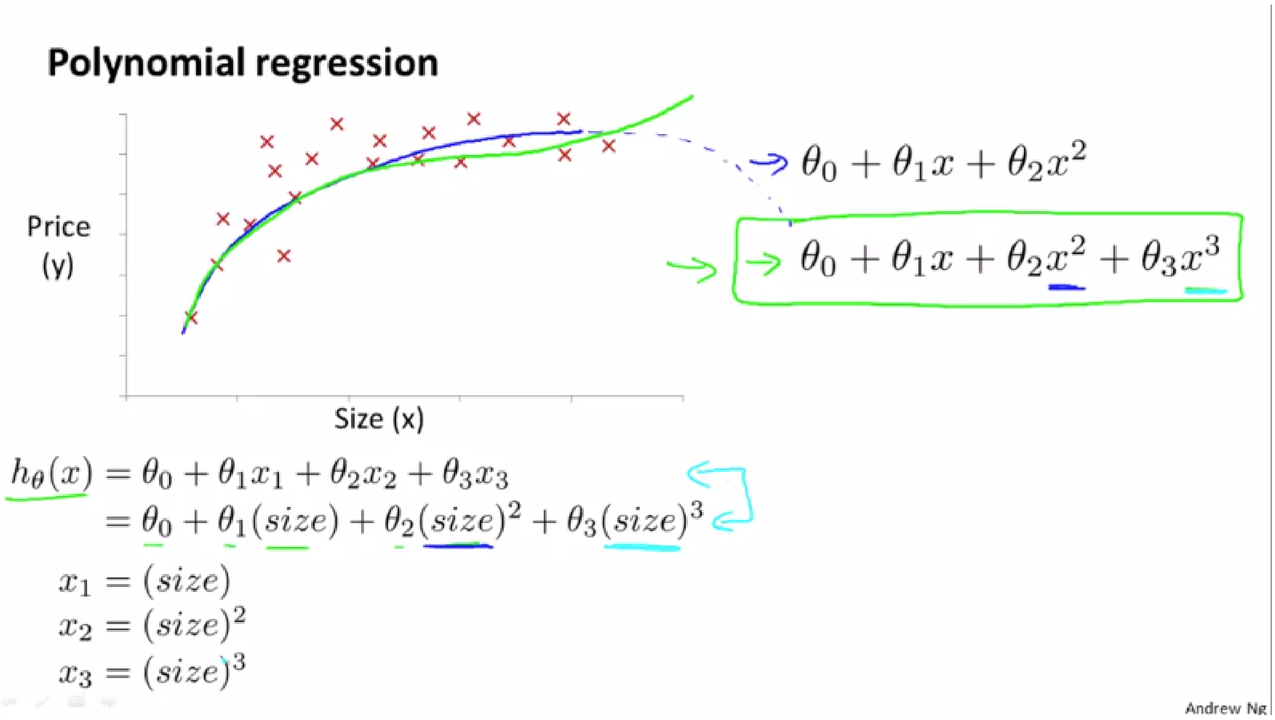
$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

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

...

Andrew Ng

3. Feature Scaling:
   a. Convergence is fast if all features are in similar scale otherwise, we might get skewed contours
   b. It’s better to scale them in 0 to 1 (or -1 to +1) which will help in reaching global minimum faster
   c. Mean normalization: subtract all values with mean (and divide by range)
4. Learning Rate (alpha) (Practical aspect):
   a. Plot minimum cost function over iterations
   b. Ideally, the plot should decrease with each iteration
   c. When the graphs flatten, it means gradient descent has converged
   d. If the graphs do not show the usual trajectory of decrement and instead increases over iterations (or looks periodic), the gradient descent is not working properly and the alpha needs to be reduced.
   e. If alpha too small, slow convergence. If alpha is large, cost function may not converge over iterations
5. Features and Polynomial Regression:
   a. We can create new features (area), instead of using the features we already have (length, width)
   b. This can help in making better model



Polynomial regression

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

$\rightarrow \boxed{\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$

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

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

Andrew Ng

c. For polynomial regression, the higher order terms (square, cube) are treated as features

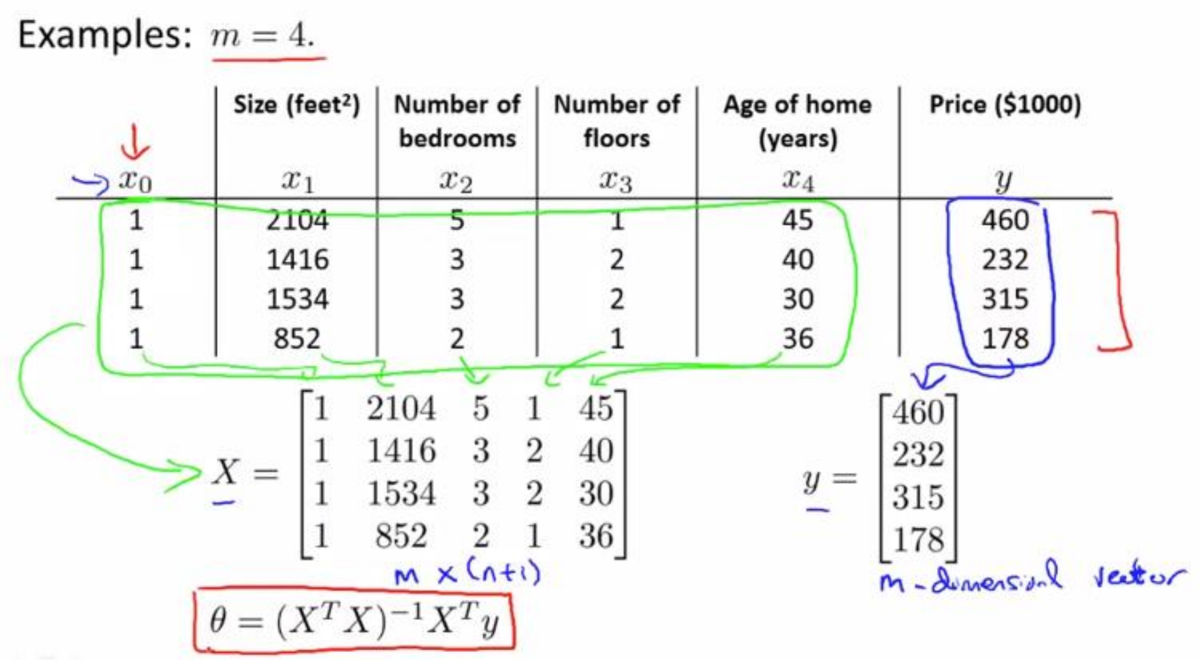
d. The range will drastically change in polynomial regression, therefore scaling is important

6. Normal Equation:

   a. It is an alternative to Gradient Descent without any need of derivation

   b. Ideal way to minimize cost function is to take derivative over all thetas, equating them to zero and thus finding all thetas (features).

   c. The way to find theta that minimizes the cost function is gives as shown below



Examples: $m = 4.$

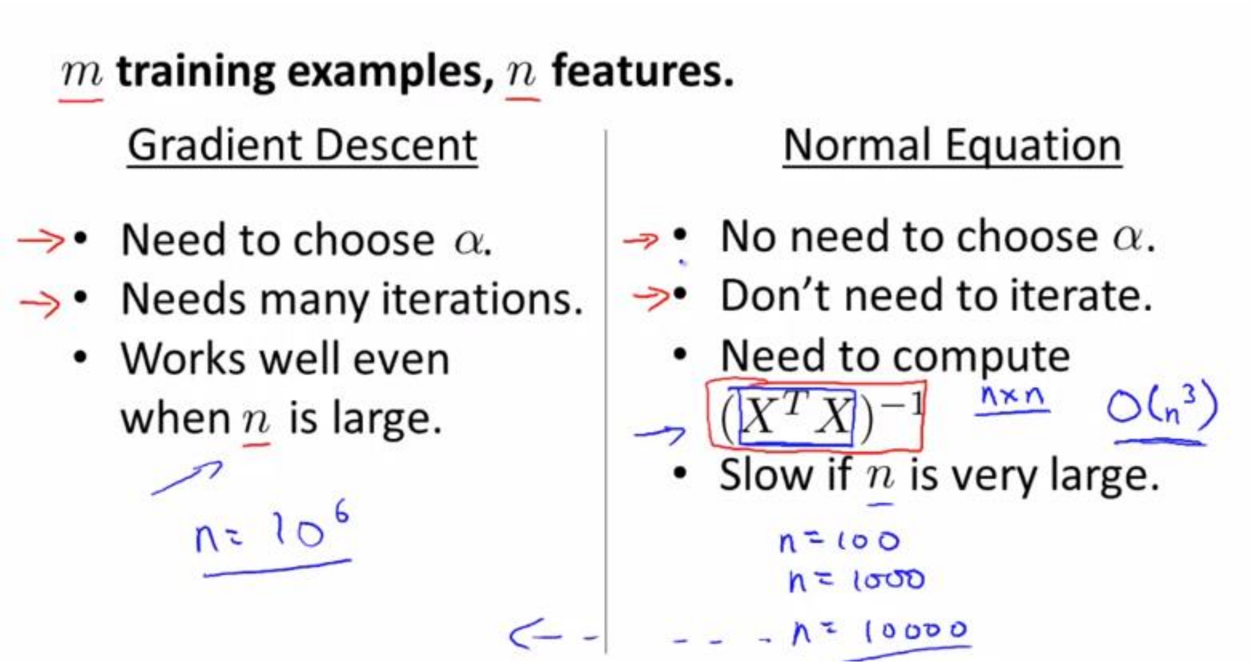
	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad y = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

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

   d. For normal equation method, it’s okay to not do feature scaling

   e. Gradient Descent v/s Normal Equation



m training examples, n features.

Gradient Descent

- Need to choose α.
- Needs many iterations.
- Works well even when n is large.

$n = 10^6$

Normal Equation

- No need to choose α.
- Don't need to iterate.
- Need to compute $\boxed{(X^TX)^{-1}}$ $n \times n$ $O(n^3)$
- Slow if n is very large.

$n = 100$

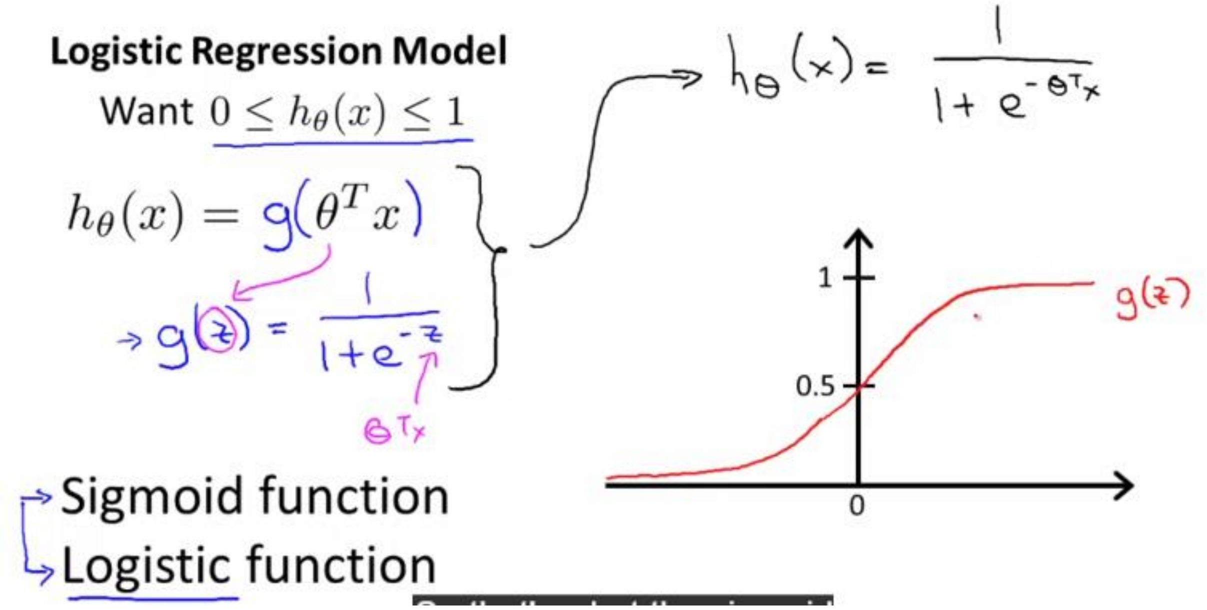
$n = 1000$

$n = 10000$

7. Normal Equation Non-invertibility (Singular): How normal equation can be implemented for singular X transpose. X
    a. It can happen if we have redundant features (columns are linearly row)
    b. Too many features (solution – delete features, use regularization)
8. MATLAB/ Octave:
    a. Octave is used extensively for prototyping
    b. Load/ Store:
        i. who/whos = show current variables
        ii. load('filename.dat') = load data from files
        iii. save filename.mat variable_name = to save variables in file (compress in binary format)
        iv. save filename.txt variable_name -ascii = human readable format
    c. Basic computation:
        i. A*B = cross product
        ii. A.*B = element wise multiplication
        iii. log(A), abs(A), exp(A)
        iv. A' = A transpose
        v. A < 3 = element wise comparison (generate 1,0 pattern)
        vi. find (A < 3) = return row and column vectors satisfying this condition
        vii. pinv(A) = inverse of A (pseudo inverse mathematically)
    d. Visualization:
        i. Imagesc() = visualize matrices

Week 3:

1. Classification:
   a. E.g., email spam, transaction fraud, tumour
   b. y (binary classification) can be 0 (negative class),1 (positive class)
   c. We can simply apply linear regression and put threshold as 0.5 to split it between 2 classes (not ideal)
2. Hypothesis - Logistic Regression:
   a. Classification algorithm (not regression like name suggests)
   b. We want hypothesis that gives answers between 0 and 1



Logistic Regression Model

Want $\underline{0 \le h_\theta(x) \le 1}$

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

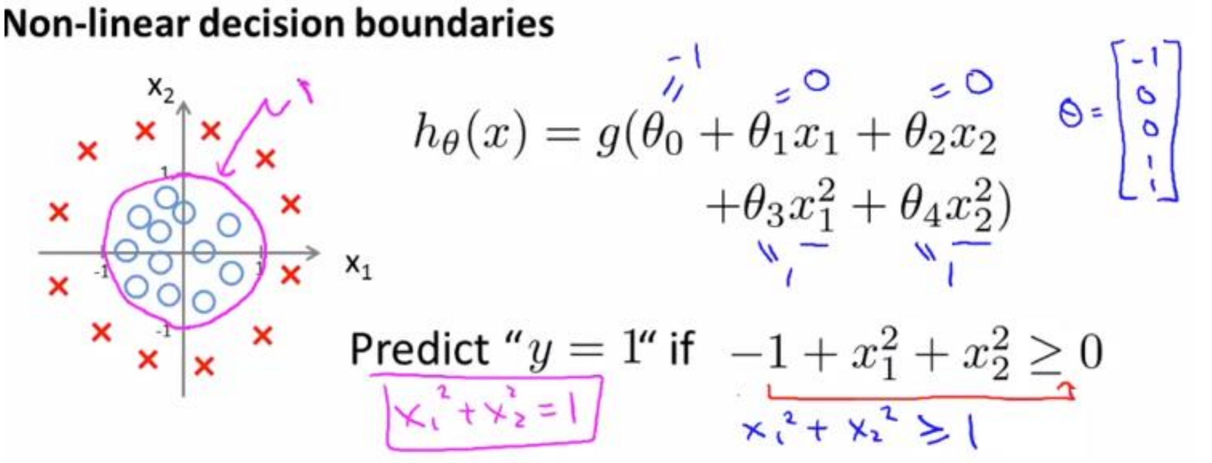
$\rightarrow g(z) = \frac{1}{1+e^{-\underset{\theta^T x}{z}}}$

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

Sigmoid function

Logistic function

   c. Sigmoid asymptotes 0 at x = -infinity and 1 at x = +infinity
   d. If h(x) = 0.7,
      i. it means there is a 70% chance (probability) of class 1 for given features x
      ii. it means there is a 30% chance (probability) of class 0 for given features x
3. Decision Boundary – Logistic Regression
   a. The 0.5 on the sigmoid is the point where we make decision for the classifier
   b. Whenever it is (theta)'*X >=0, it is class 1 otherwise class 0 (this line is the decision boundary)
   c. Decision boundary is the property of the hypothesis and not of the dataset (need dataset for fitting)
   d. We can add higher order terms to develop for non-linear decision boundaries



Non-linear decision boundaries

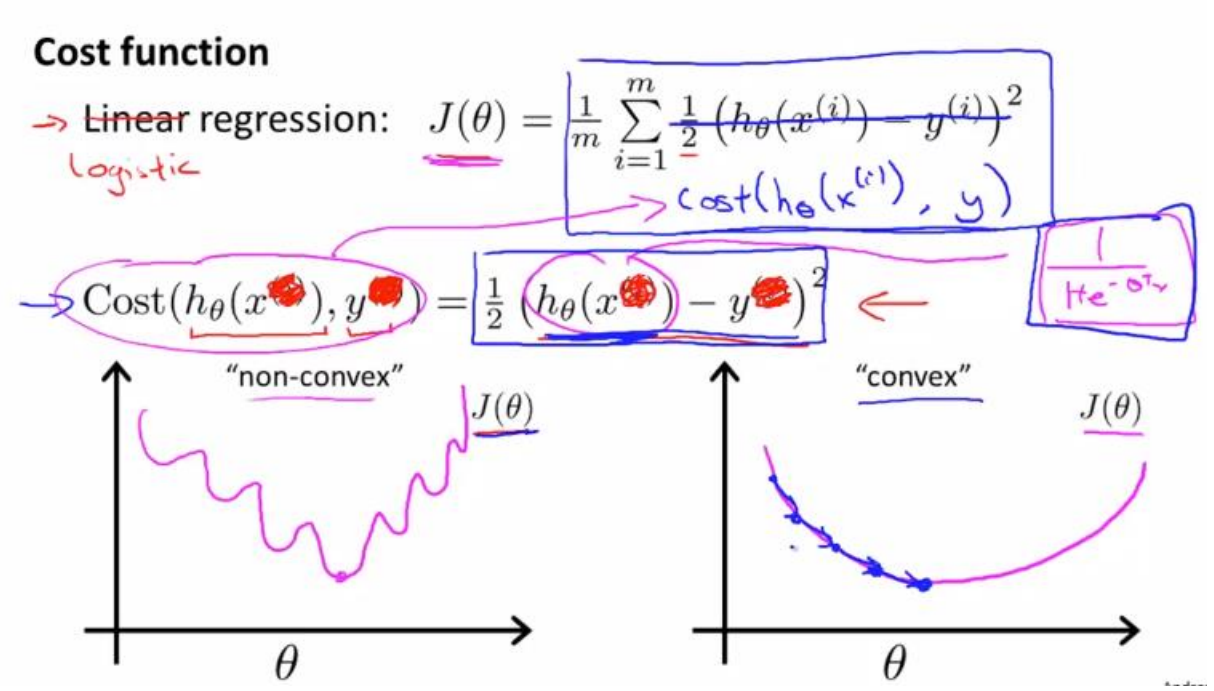
$$h_\theta(x) = g(\overset{-1}{\theta_0} + \overset{0}{\theta_1} x_1 + \overset{0}{\theta_2} x_2 + \underset{1}{\theta_3} x_1^2 + \underset{1}{\theta_4} x_2^2) \qquad \theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $\underbrace{-1 + x_1^2 + x_2^2 \ge 0}_{x_1^2 + x_2^2 \ge 1}$

$\boxed{x_1^2 + x_2^2 = 1}$

4. Cost function: optimization function to fit the hypothesis parameters
   a. If we use the cost (squared error) from linear regression for logistic regression, the cost function will be non-convex meaning that it will have multiple local minima and therefore there is no guarantee to reach global minima by gradient descent (by plugging sigmoid in linear regression cost equation)

b. To overcome this, we need J(theta) to be a convex function of theta despite having a non-linear term of sigmoid in equation of hypothesis, which squared error does not allow



Cost function

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

Logistic

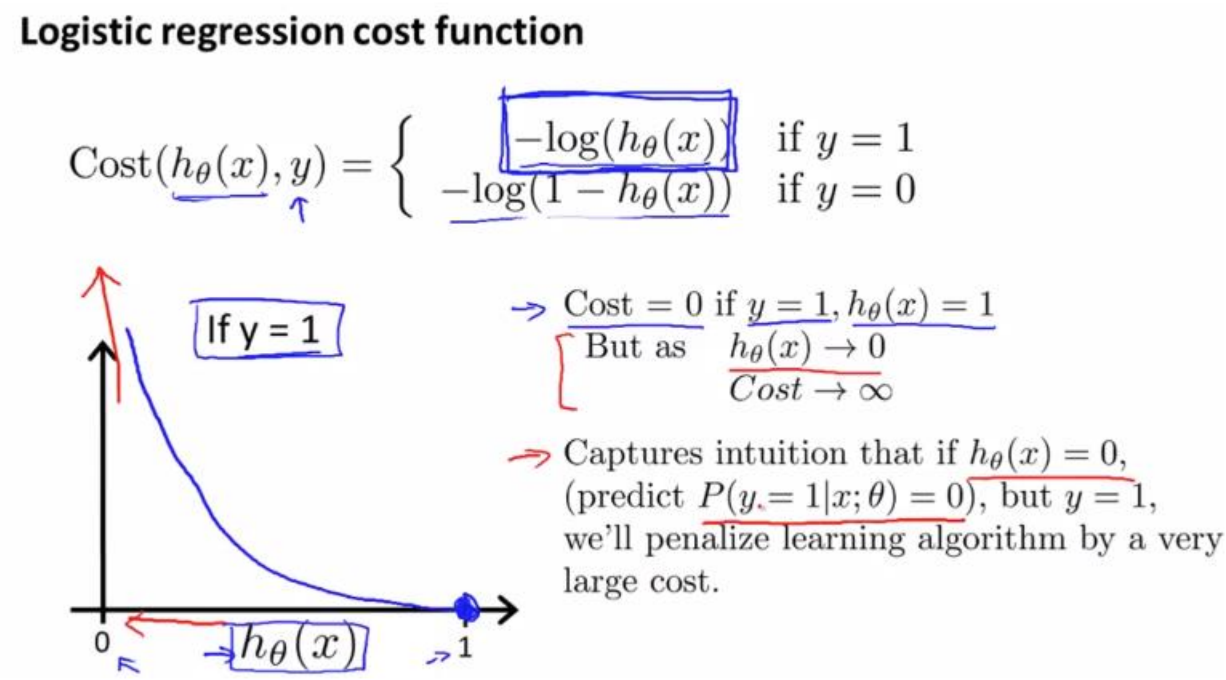
→ $\mathrm{cost}(h_\theta(x^{(i)}), y)$

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

$\frac{1}{1+e^{-\theta^T x}}$

c. Cost equation must have a logarithmic term (unlike squared error in linear regression) so as to compensate for the non-linear sigmoid hypothesis and therefore to eventually have a convex cost function.

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

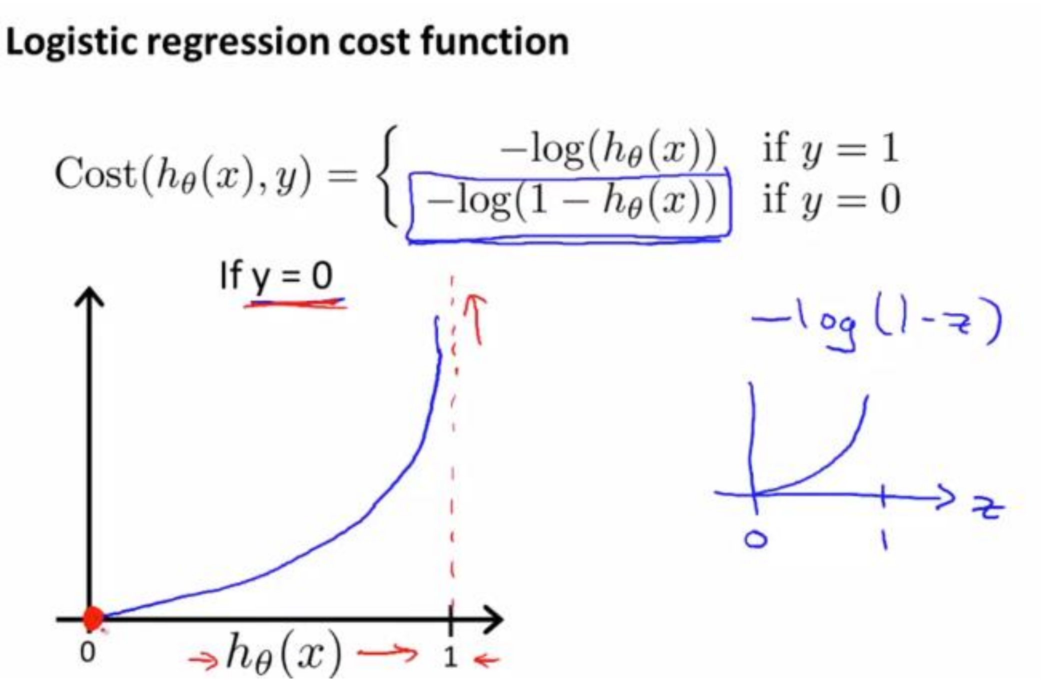


Logistic regression cost function

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

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

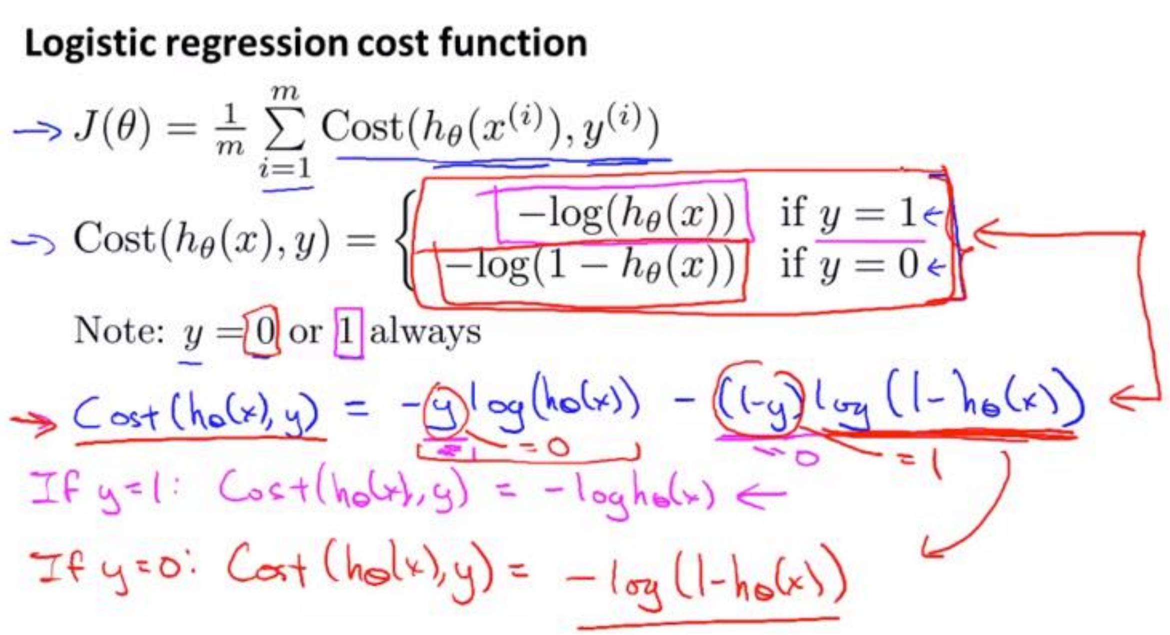
→ Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x;\theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.



Logistic regression cost function

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

5. Simplified cost function and gradient descent:
   a. Combining the cost function equation as shown below:



Logistic regression cost function

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

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

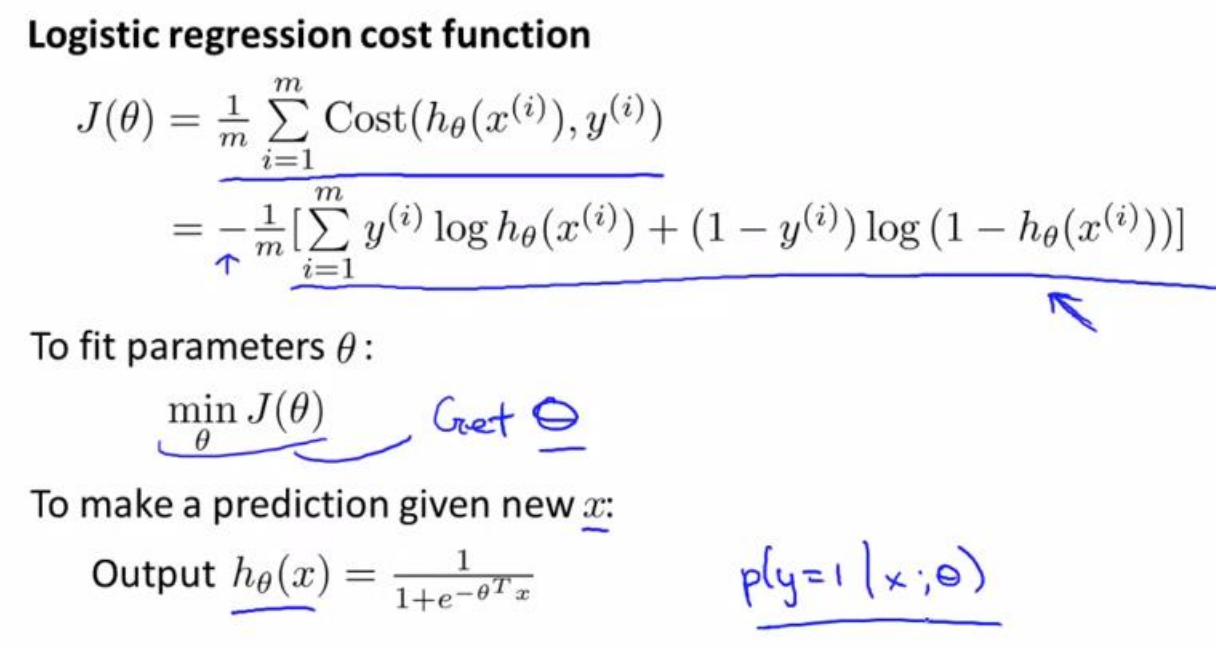
Note: $y = \boxed{0}$ or $\boxed{1}$ always

$$\rightarrow \mathrm{Cost}(h_\theta(x),y) = -\underbrace{y}_{=1}\underbrace{\log(h_\theta(x))}_{=0} - \underbrace{(1-y)}_{=0}\underbrace{\log(1-h_\theta(x))}_{=1}$$

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

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

   b. This particular cost function has been selected from a statistical perspective of Maximum Likelihood Estimation which helps in selection theta for different models and is convex



Logistic regression cost function

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

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

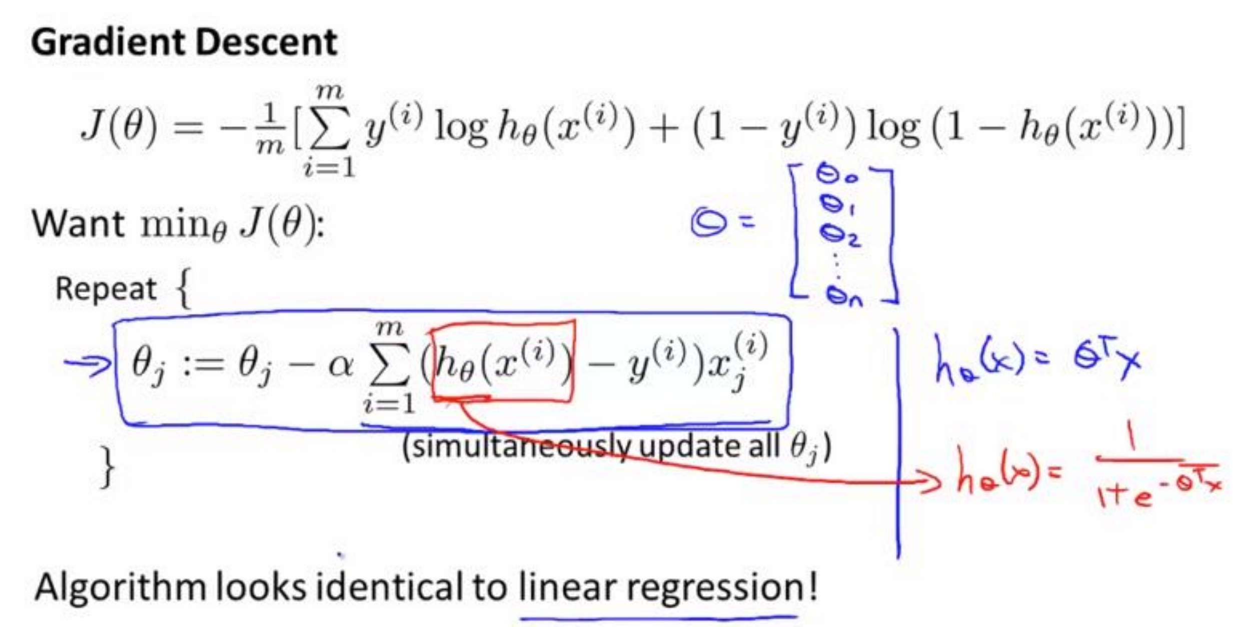
To fit parameters θ:

$$\min_\theta J(\theta)$$ Get Θ

To make a prediction given new x:

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

c. Minimization of J(theta) will help us find optimum values of theta to develop the hypothesis and it will be done by gradient descent as shown. The derivative of J when put into the theta update formula looks exactly like linear regression, the difference between the two is the hypothesis function



Gradient Descent

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

Want $\min_\theta J(\theta)$: $\Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$

Repeat {

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

(simultaneously update all θ_j)

}

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

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

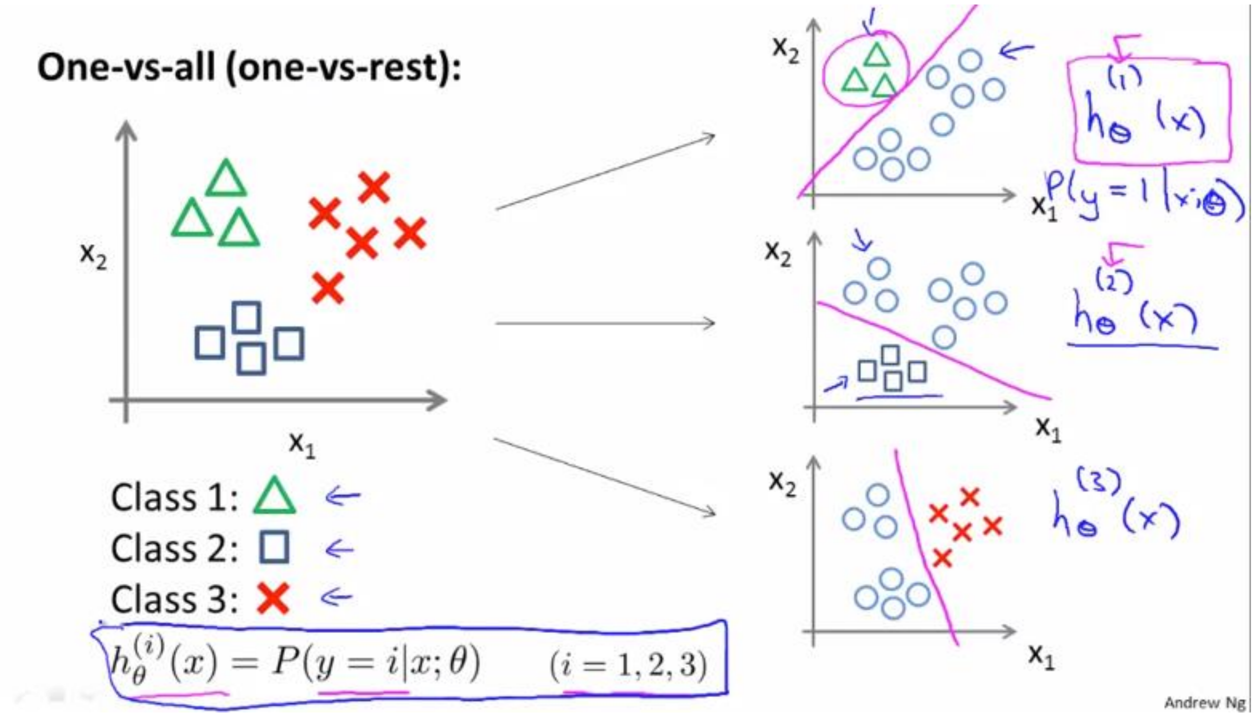
Algorithm looks identical to linear regression!

d. For proper convergence, plot J as a function of number of iterations (like in linear regression) and check if it is decreasing or not.

e. We can apply feature scaling like in linear regression to make convergence faster

6. Advanced Optimization:
   a. The concept of optimization of parameters involves computation of cost function and its derivative
   b. Some sophisticated algorithms (other than gradient descent) are: Conjugate gradient, BFGS and L-BFGS. They are much complex (disadvantage) but we do not need to pick learning rate manually and also, they are much faster than gradient descent (advantage)
   c. For practical implementation:
      i. Define your own cost function which takes input as theta vector and outputs value of J and gradient for each theta (no of parameters (theta) X 1 vector)
      ii. Pass them to optimset function, fminunc for their optimal values
7. Multiclass classification: One v/s all algorithm
   a. If you have N classes, divide them into N different binary classification problems
   b. Then, train a logistic regression model and make a decision boundary for one particular class (v/s all)
   c. Eventually we will have N different equations for h(x) for each of the class

d. While making prediction we will assign the class corresponding to the maximum value of h(theta)



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

Class 1: △ ←

Class 2: □ ←

Class 3: ✕ ←

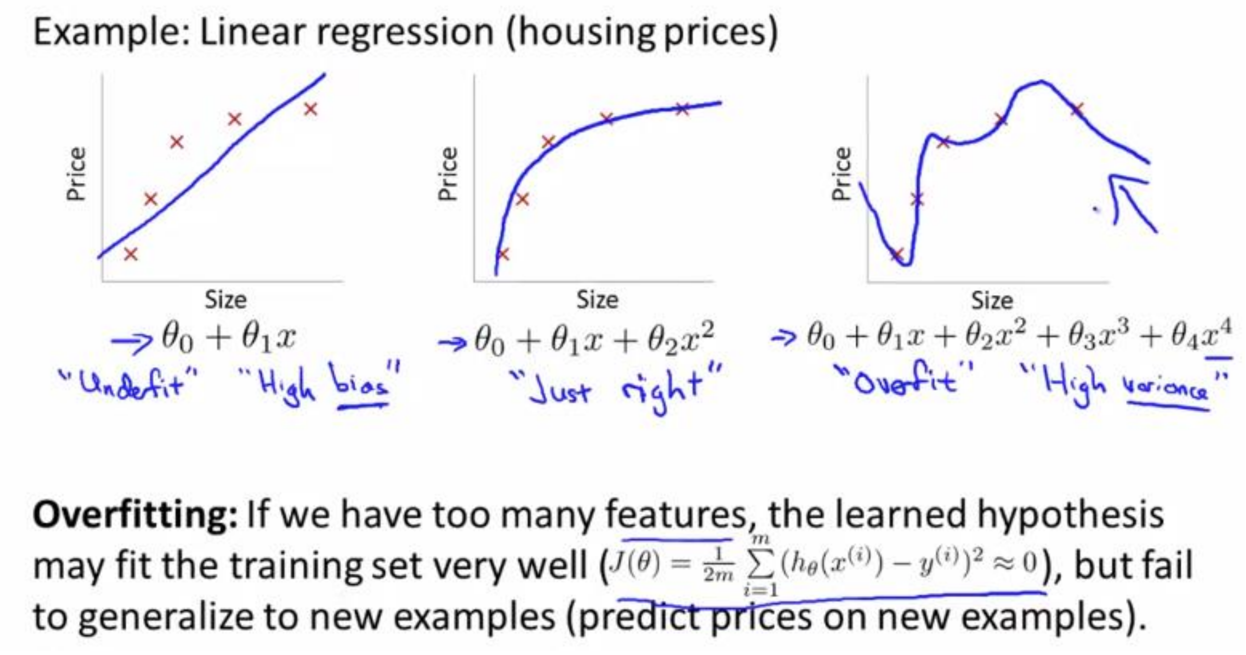
$$\boxed{h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)}$$

8. Overfitting:

a. Problem is overfitting, can be solved/ ameliorated by regularization

b. Underfitting means that the hypothesis does not fit the training very well and has high bias

c. Overfitting means that the hypothesis has high variance, variance is too much, not enough data to fit



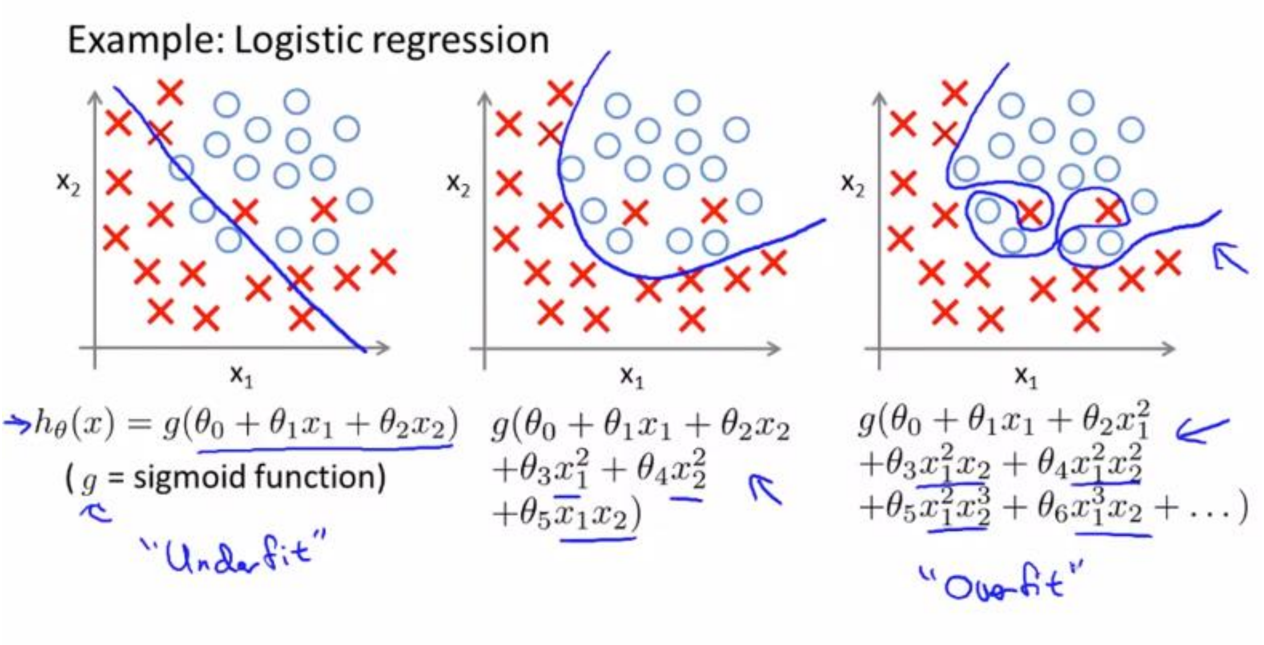
Example: Linear regression (housing prices)

$\rightarrow \theta_0 + \theta_1 x$ "Undefit" "High bias"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$ "Just right"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ "Overfit" "High variance"

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



Example: Logistic regression

$h_\theta(x) = g(\theta_0+\theta_1x_1+\theta_2x_2)$

(g = sigmoid function)

"Underfit"

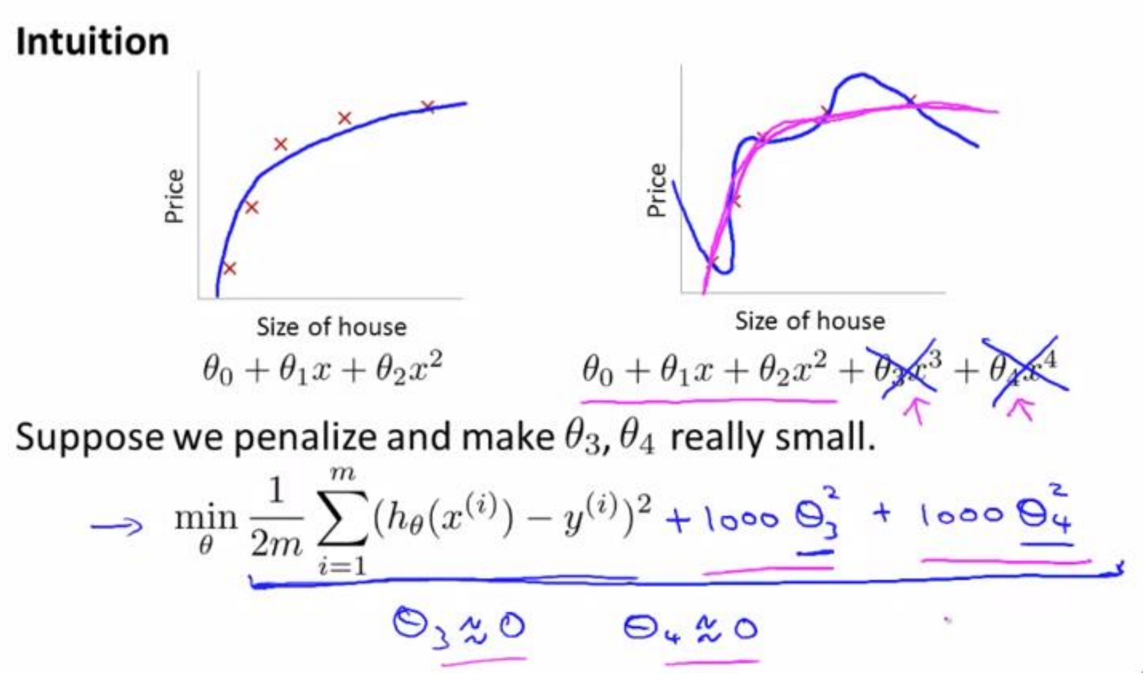
$g(\theta_0+\theta_1x_1+\theta_2x_2$ $+\theta_3x_1^2+\theta_4x_2^2$ $+\theta_5x_1x_2)$

$g(\theta_0+\theta_1x_1+\theta_2x_1^2$ $+\theta_3x_1^2x_2+\theta_4x_1^2x_2^2$ $+\theta_5x_1^2x_2^3+\theta_6x_1^3x_2+\dots)$

"Overfit"

d. Addressing overfitting:
   i. Reduce number of features
      1. Manually remove features
      2. Model selection algorithm
   ii. Regularization
      1. Keep all features but reduce magnitude/value of parameter theta

9. Regularization:
   a. We achieve it by making the higher order thetas very small close to zero so as to not have their effect



Intuition

$\theta_0+\theta_1x+\theta_2x^2$

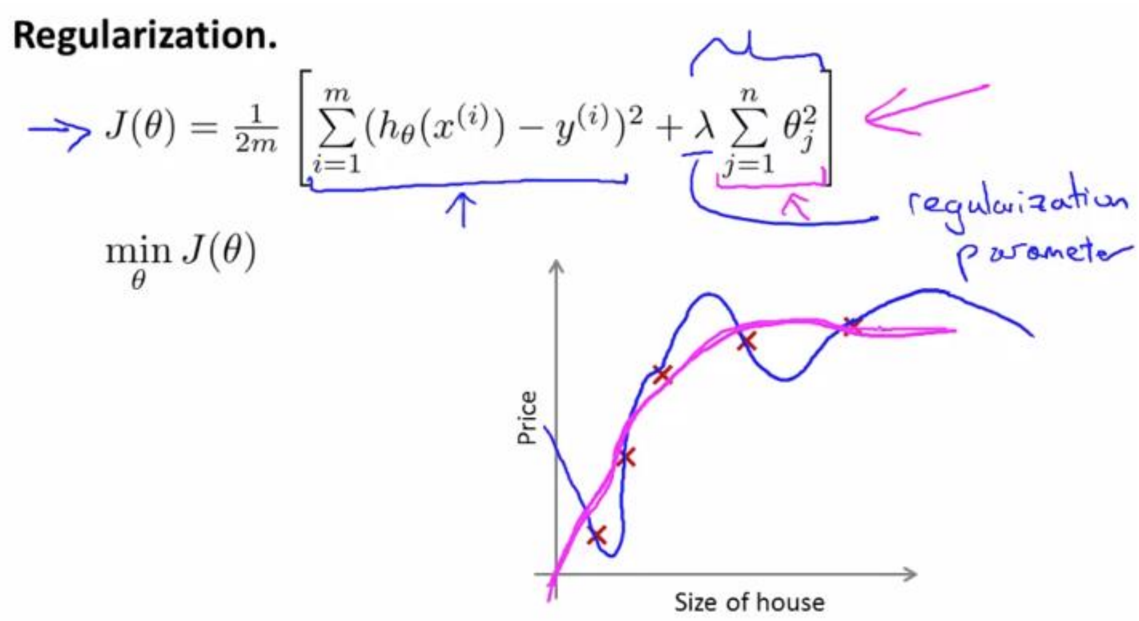
$\theta_0+\theta_1x+\theta_2x^2+\cancel{\theta_3x^3}+\cancel{\theta_4x^4}$

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2 + 1000\,\Theta_3^2 + 1000\,\Theta_4^2$$

$\Theta_3 \approx 0$ $\Theta_4 \approx 0$

   b. Idea – having smaller values of theta will result in developing simpler hypothesis and thus smoother
   c. We add a regularization term to the cost function equation
   d. Lambda does the job of keeping the parameters small → simple hypothesis → avoid overfitting



Regularization.

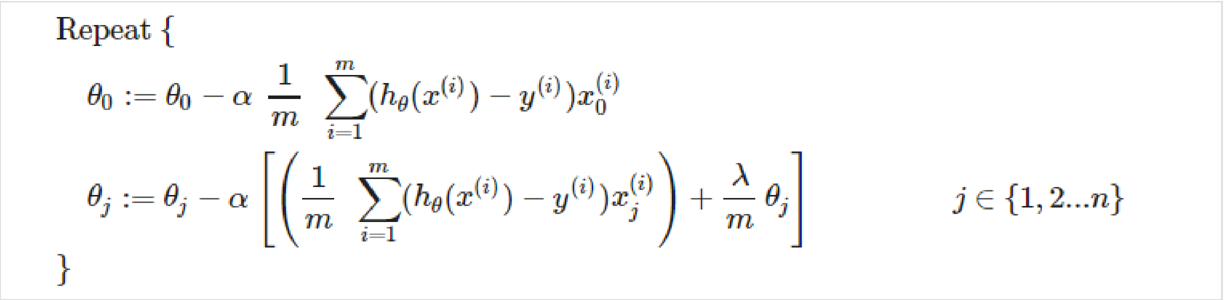
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

regularization parameter

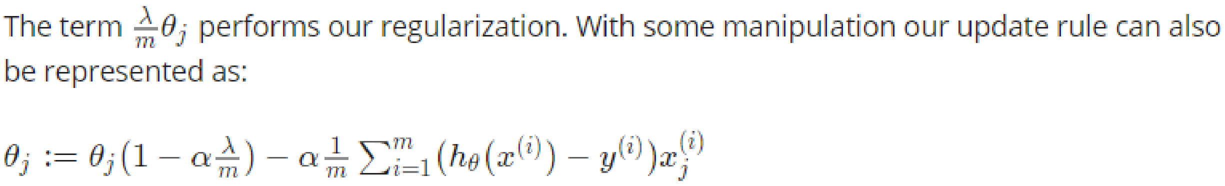
$$\min_\theta J(\theta)$$

   e. Very large lambda will underfit the hypothesis because of very high bias
   f. Therefore, the choice of lambda (regularization parameter) is crucial

10. Regularized linear regression:
   a. Using new definition of J(theta), we will find a new derivative term to find theta update equations



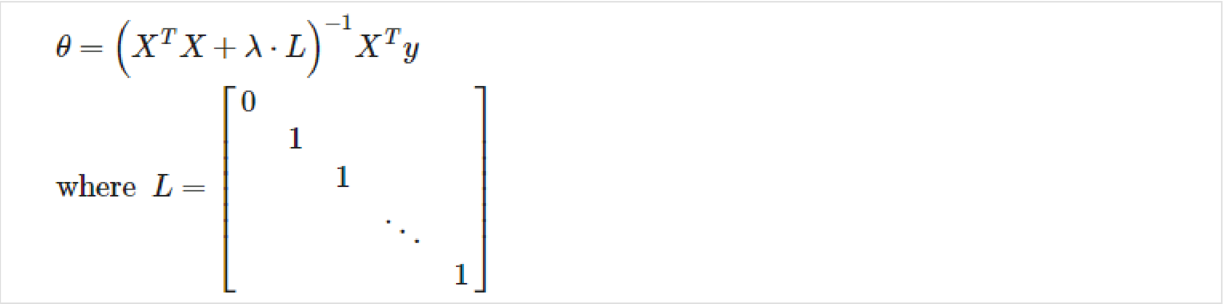
$$\begin{aligned}&\text{Repeat } \{\\ &\quad \theta_0 := \theta_0 - \alpha\ \frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}\\ &\quad \theta_j := \theta_j - \alpha\left[\left(\frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right] \qquad\qquad j \in \{1, 2...n\}\\ &\}\end{aligned}$$



The term $\frac{\lambda}{m}\theta_j$ performs our regularization. With some manipulation our update rule can also be represented as:

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

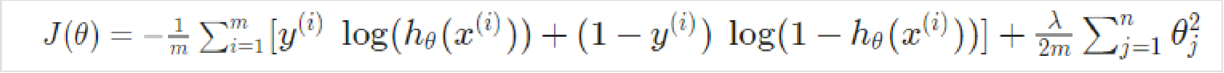
   b. Modified normal equation:



$$\theta = \left(X^TX + \lambda \cdot L\right)^{-1}X^Ty$$

$$\text{where } L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

11. Regularized Logistic Regression:
   a. Very high order polynomials added to the hypothesis can cause overfitting in logistic regression



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

Gradient descent

Repeat $\{$

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

$$\rightarrow \quad \theta_j := \theta_j - \alpha \underbrace{\left[\frac{1}{m} \sum_{i=1}^{m} (\underline{h_\theta(x^{(i)})} - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]}_{\frac{\partial}{\partial \theta_j} \underline{J(\theta)}} \leftarrow$$

$$(j = \times \underset{\theta_1 \cdots\cdots \theta_n}{\underline{1, 2, 3, \ldots, n}})$$

$\}$

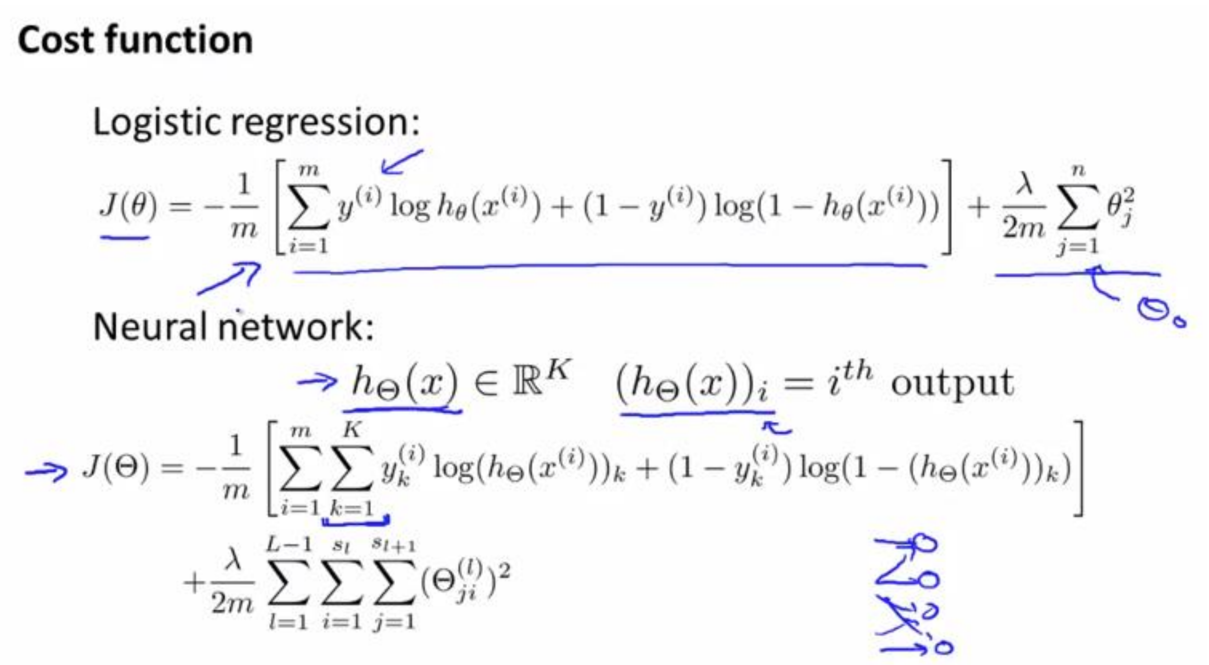
$$\underline{h_\theta(x)} = \frac{1}{1 + e^{-\theta^T x}}$$

Week 4:

1. Non-linear hypothesis – Neural Network:
    a. One of the ways to have non-linearity in classification is to have higher order polynomials in logistic regression. But for large number of features, including higher order polynomials would be unfeasible eventually making the model overfit and also would be computationally expensive
    b. For sophisticated applications like CV, linear hypothesis can not be used thus the need for NN
2. Neuron and the Brain:
    a. Most algorithms mimic something, therefore why not go for the best and mimic the human brain
    b. Because of the neuro-rewiring experiments, it is concluded that there can be one algorithm which can work well for a large range of problems
    c. Examples (plug in any sensor and allow brain to learn):
        i. Seeing with your tongue
        ii. Human echolocation
        iii. Haptic belt: direction sense
        iv. Implanting a third eye
3. Neural Network representation:
    a. Dendrites are inputs, axon is output, nucleus does computation on input
    b. In NN terminology, theta which are parameters of hypothesis are called weights
    c. The first layer is input layer, last layer is output layer and middle are hidden layers
    d. Forward propagation is used to compute output of NN (use vectorized method for faster implement)
    e. Architecture of NN is the way in which they are connected
4. Examples:
    a. XOR/ XNOR problem (non-linear classification problem)
    b. AND/ OR/ NOT are linear function and therefore can be realized by 1 neuron (or single layer NN)
5. Multiclass classification:

Week 5:

1. Cost Function for NN:
   a. L denotes total number of layers and s denotes number of units in each layer
   b. Cost function will be generalized version of regularized logistic regression



Cost function

Logistic regression:

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

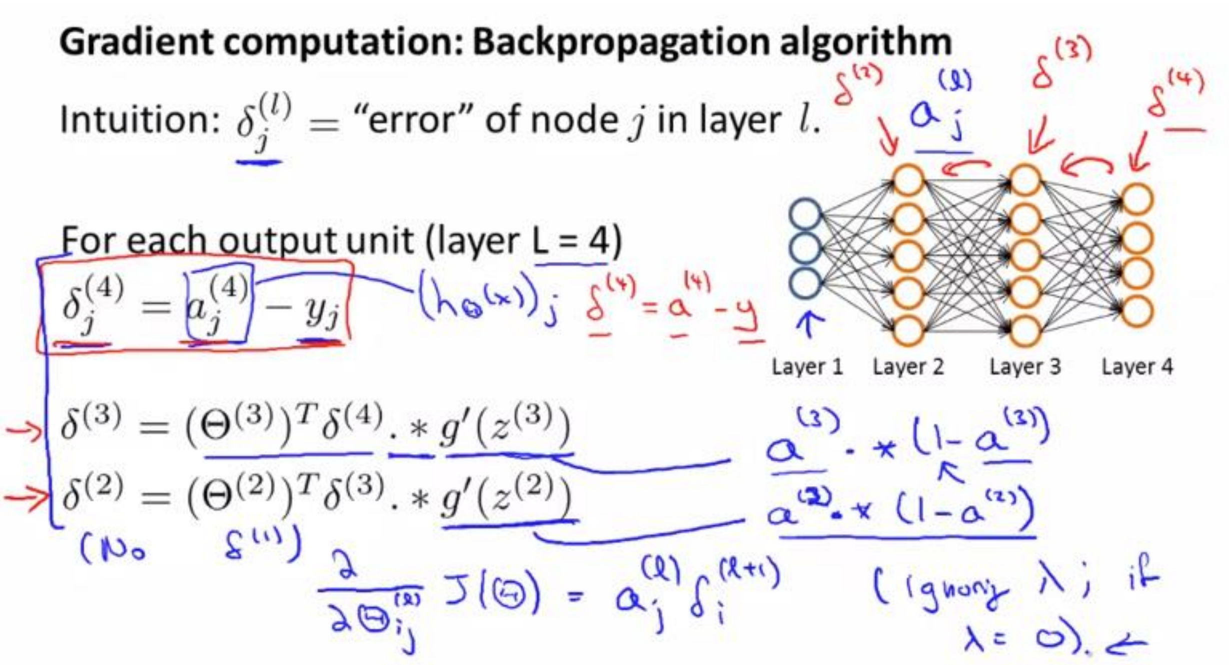
θ_0

Neural network:

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

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

2. Backpropagation:
   a. Delta term calculates error at all nodes in each layer



Gradient computation: Backpropagation algorithm

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

$\delta^{(2)}$ $a_j^{(l)}$ $\delta^{(3)}$ $\delta^{(4)}$

For each output unit (layer L = 4)

$$\boxed{\delta_j^{(4)} = \boxed{a_j^{(4)}} - y_j} \quad (h_\Theta(x))_j \quad \delta^{(4)} = a^{(4)} - y$$

$$\delta^{(3)} = (\Theta^{(3)})^T\delta^{(4)} .* g'(z^{(3)}) \qquad a^{(3)} .* (1 - a^{(3)})$$

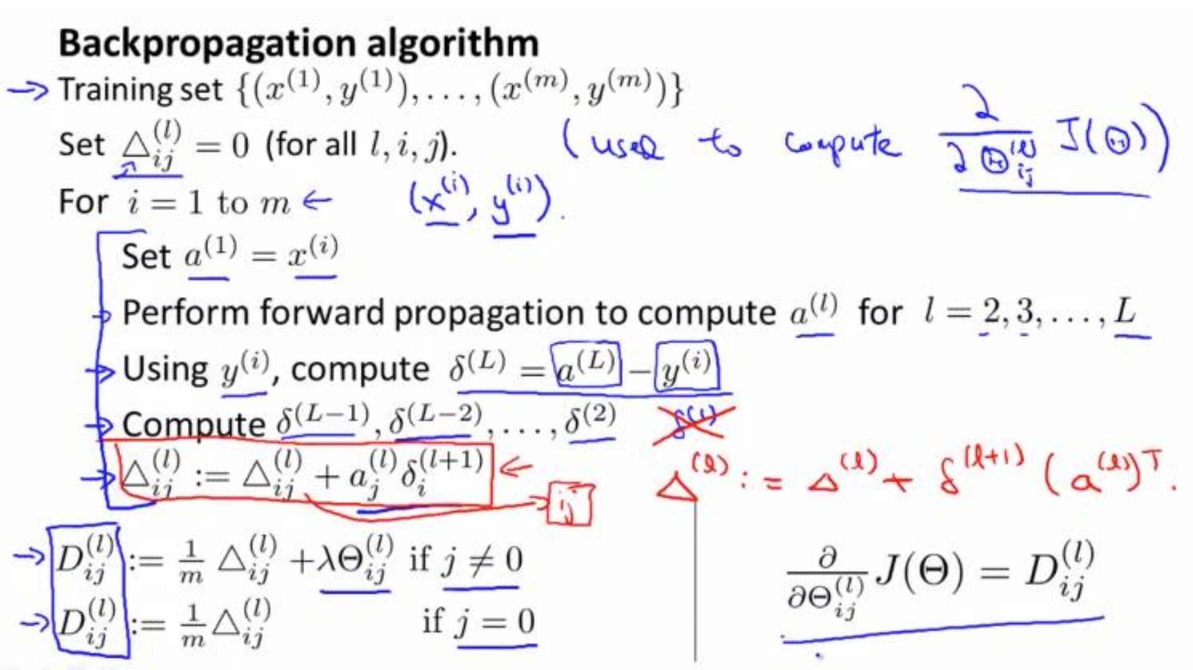
$$\delta^{(2)} = (\Theta^{(2)})^T\delta^{(3)} .* g'(z^{(2)}) \qquad a^{(2)} .* (1 - a^{(2)})$$

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

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

(ignoring λ; if $\lambda = 0$).

   b. Algorithm:



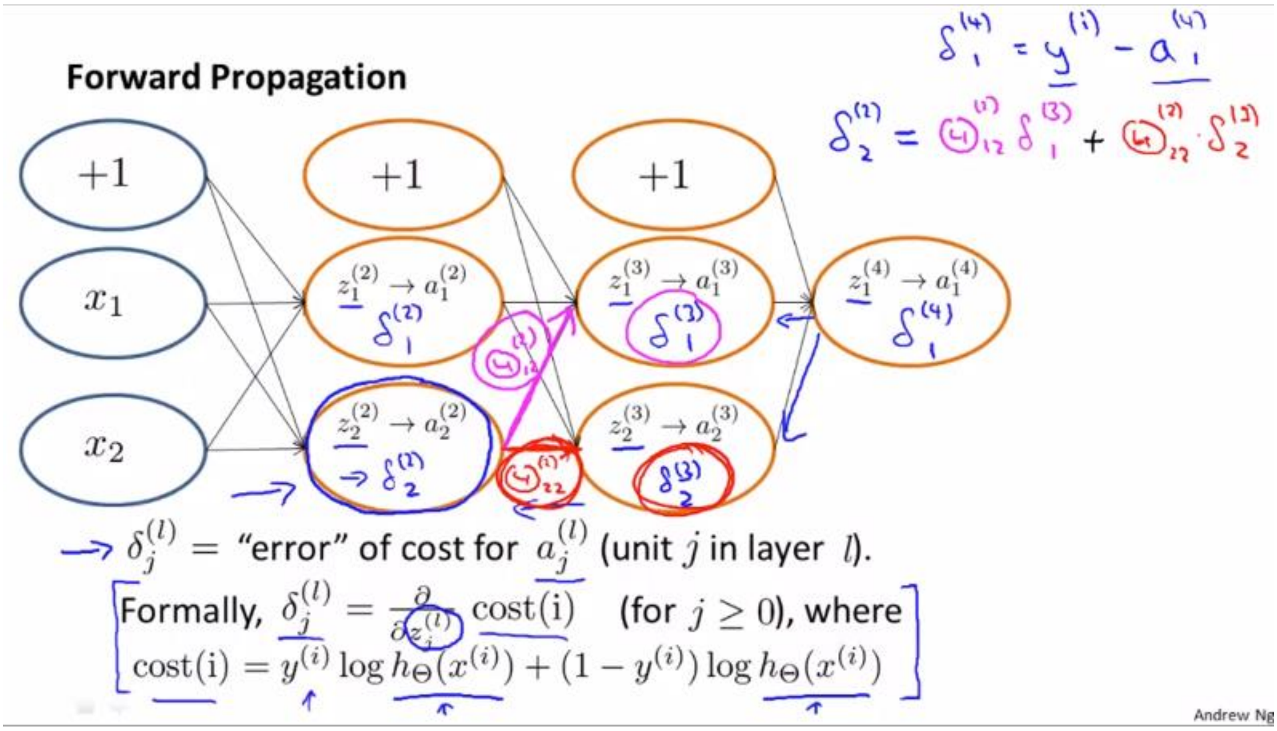
Backpropagation algorithm

```
Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
Set $\Delta_{ij}^{(l)} = 0$ (for all $l, i, j$).   ( use to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ )
For $i = 1$ to $m$ ←   $(x^{(i)}, y^{(i)})$.
    Set $a^{(1)} = x^{(i)}$
    Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$
    Using $y^{(i)}$, compute $\delta^{(L)} = \boxed{a^{(L)}} - \boxed{y^{(i)}}$
    Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$   $\cancel{\delta^{(1)}}$
    $\boxed{\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}}$ ←   $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$.
$\boxed{D_{ij}^{(l)}} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$   if $j \neq 0$
$\boxed{D_{ij}^{(l)}} := \frac{1}{m} \Delta_{ij}^{(l)}$   if $j = 0$
```

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

c. Back Propagation intuition:

   i. For intuition, the delta terms can be calculated similar to the FP by multiplying weights and output to generate corresponding delta at input side.



Forward Propagation

$\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$

$\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$

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

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

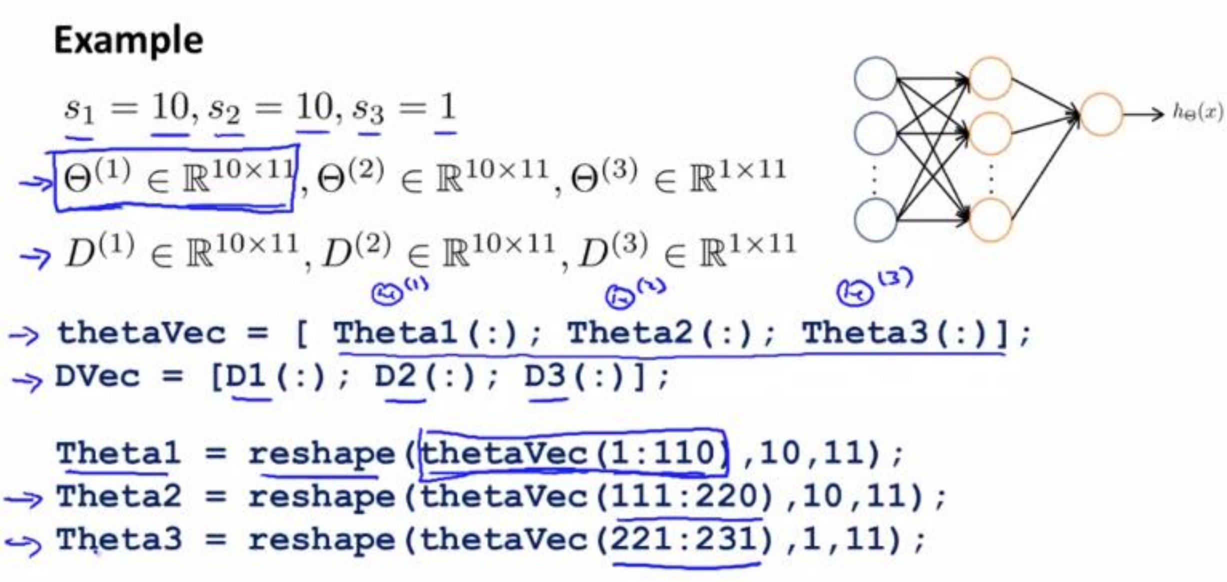
$\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$

Andrew Ng

d. Unrolling parameters:

   i. Earlier we passed theta vectors to optimization functions like fminunc, now our parameters are matrices

   ii. Method to make matrix to vector and vice versa so that it is compatible with optimization function



Example

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

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

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

$\Theta^{(1)}$ $\Theta^{(2)}$ $\Theta^{(3)}$

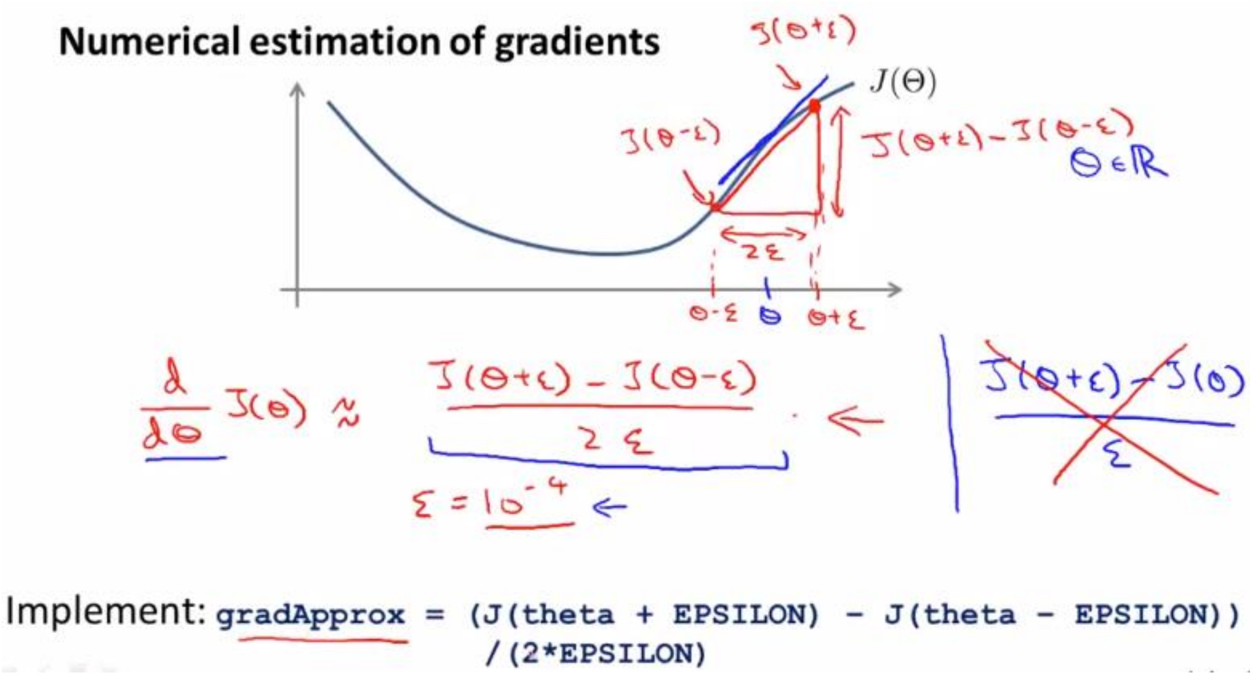
```
thetaVec = [ Theta1(:); Theta2(:); Theta3(:)];
DVec = [D1(:); D2(:); D3(:)];

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

e. Gradient Checking:

i. Gradient checking helps in accurate implementation of forward and back prop for complicated neural networks

ii. Method to compute approximate gradient is as shown



Numerical estimation of gradients

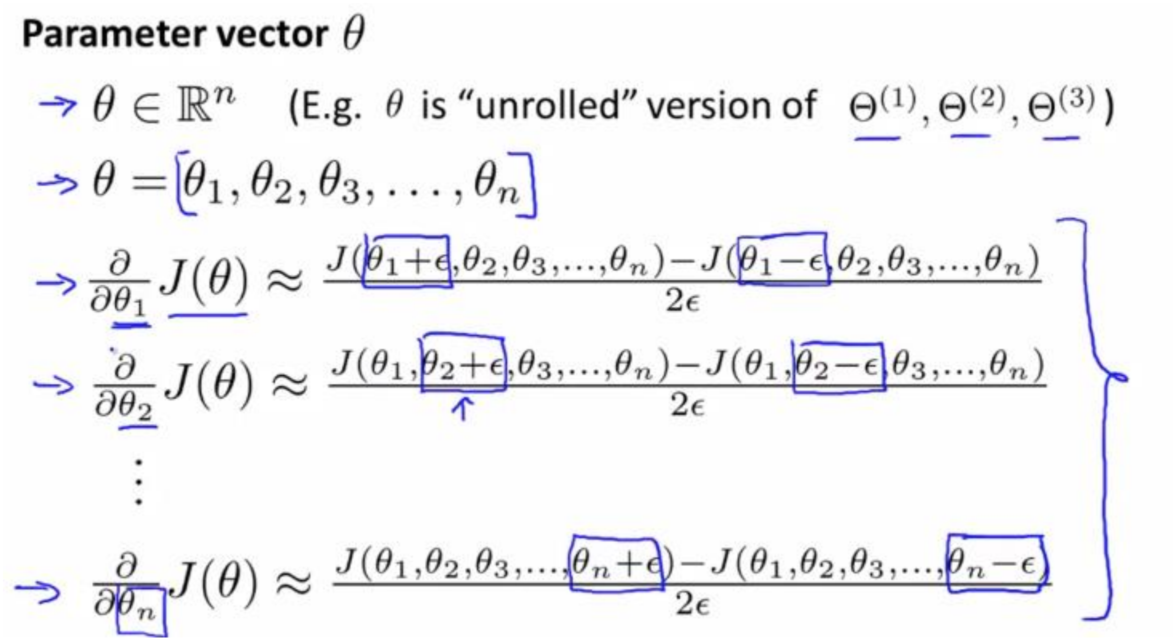
$$\frac{d}{d\theta}J(\theta) \approx \underbrace{\frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}}$$

$\epsilon = 10^{-4}$

$$\cancel{\frac{J(\theta+\epsilon)-J(\theta)}{\epsilon}}$$

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

iii. We can use this method of numeric approximation can be used to calculate partial derivative of cost function with respect to all thetas as shown. We can compare the actual gradient of back propagation and these numerically computed ones and must be almost equal.



Parameter vector θ

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

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

$$\left.\begin{aligned}
&\rightarrow \frac{\partial}{\underline{\partial\theta_1}} \underline{J(\theta)} \approx \frac{J(\boxed{\theta_1+\epsilon}, \theta_2, \theta_3, \ldots, \theta_n) - J(\boxed{\theta_1-\epsilon}, \theta_2, \theta_3, \ldots, \theta_n)}{2\epsilon} \\
&\rightarrow \frac{\partial}{\underline{\partial\theta_2}} J(\theta) \approx \frac{J(\theta_1, \underset{\uparrow}{\boxed{\theta_2+\epsilon}}, \theta_3, \ldots, \theta_n) - J(\theta_1, \boxed{\theta_2-\epsilon}, \theta_3, \ldots, \theta_n)}{2\epsilon} \\
&\qquad \vdots \\
&\rightarrow \frac{\partial}{\partial\boxed{\theta_n}} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \ldots, \boxed{\theta_n+\epsilon}) - J(\theta_1, \theta_2, \theta_3, \ldots, \boxed{\theta_n-\epsilon})}{2\epsilon}
\end{aligned}\right\}$$

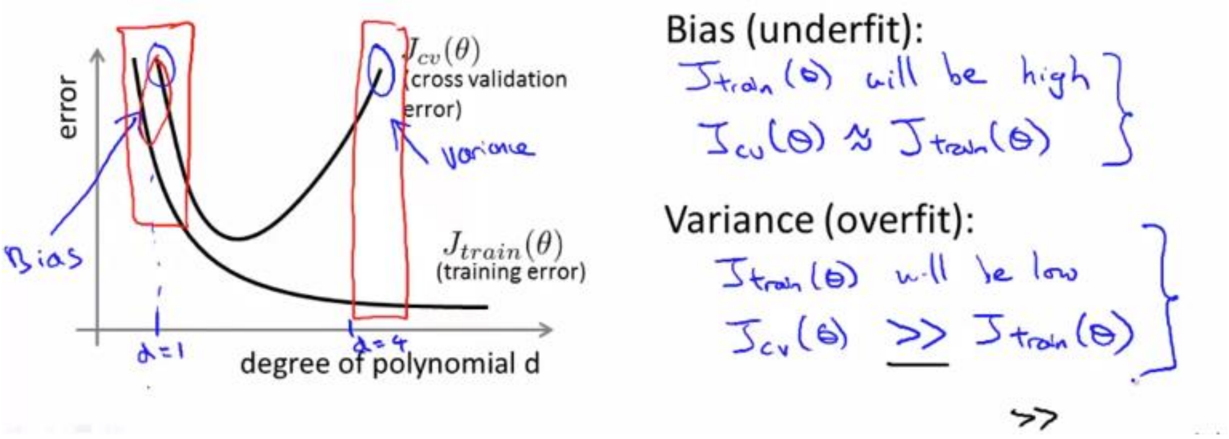
f. Random Initialization:
   i. We can't initialize thetas as zeros for NN because then all neurons of hidden layers will act as identical
   ii. To break this symmetry, we implement random initialization

g. Steps:
   i. Chose architecture (no layers, i/p, o/p, etc)
      1. No of input = feature dimensions
      2. No of output = no of classes in multiclass classification
      3. Hidden = take 1 (common), have same no. of neurons in hidden (usually)
   ii. Randomly initialize weights
   iii. Implement forward propagation to get h for any x
   iv. Implement cost function to calculate J
   v. Implement backpropagation to compute partial derivation (iterate over examples)
      1. Get activation and delta terms
      2. Get del and partial derivative terms (with regularization)
   vi. Use gradient checking to compare backprop and approx. numeric estimation method
   vii. Use gradient descent/ optimization method along with backprop to minimize J

3. Example of NN:
   a. Autonomous driving
   b. Training car by driving it
   c. Every 2 sec footage is used as training image and used as input for NN
   d. After 2min of training, the algo learns accurately to mimic driver
   e. To validate, 12 image/ sec sampling is done and multiple NN are used to give decisions and most confident network results are used to control the vehicle

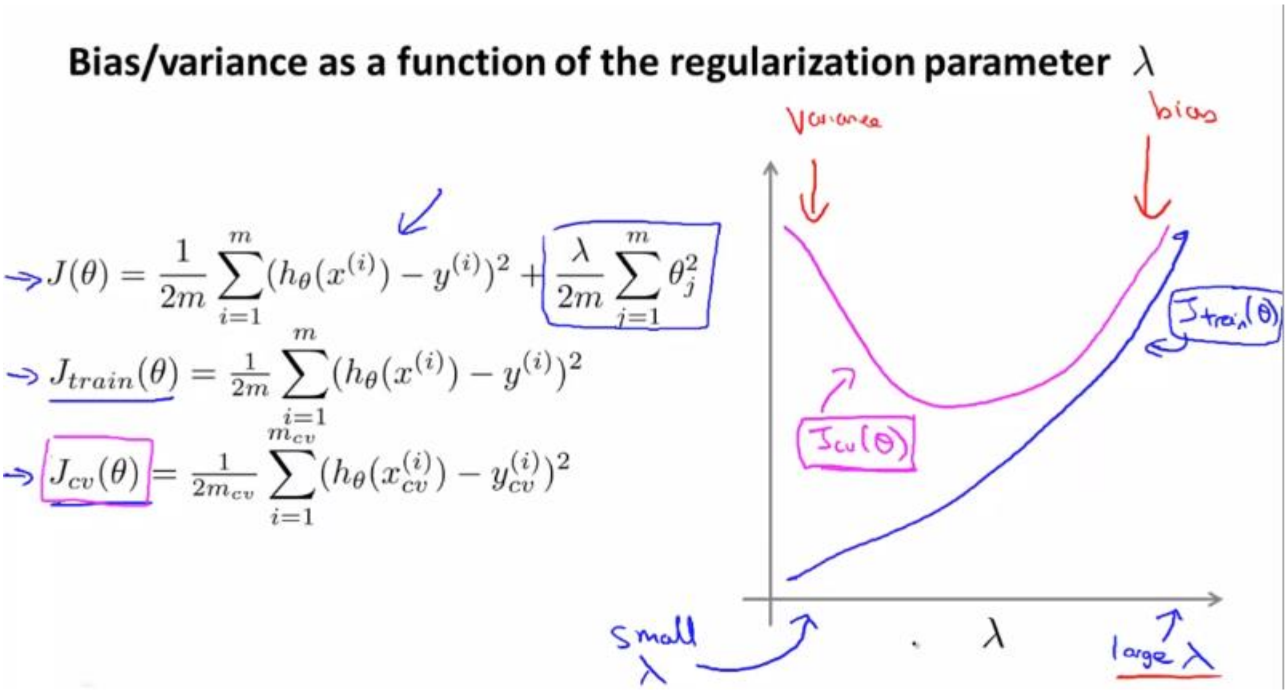
Week 6:

1. Applied ML/ Practical Guidelines: If u implemented a regularized linear regression model and our hypothesis gave large errors in its prediction
    a. Get more training examples (helps in high variance problem)
    b. Trying smaller set of features (helps in high variance problem)
    c. Get additional features (helps in high bias problem)
    d. Adding polynomial terms (helps in high bias problem)
    e. Increasing lambda (helps in high variance problem)
    f. Decreasing lambda (helps in high bias problem)
2. Machine Learning diagnostic:
    a. Evaluation of hypothesis:
        i. Low error does not necessarily mean good hypo (case of overfitting)
        ii. Plotting and observing hypo function is not be feasible for large dimensions
        iii. Solution can be (same for classifier/ regression model):
            1. Split dataset into 70%, 30% splits
            2. Use 70% data to build model
            3. Also sort data randomly (shuffle) and then feed for training/ testing
            4. Find Train error J theta (70% data), also find test set error (30% data)
            5. Misclassification error is used for as error function for classifiers.
    b. Model selection and train/ validation/ test set:
        i. Training error not a good measure of how good model will hypothesis be (overfitting)
        ii. Model selection problem is confusion on what degree of polynomial to keep in hypothesis
        iii. Process:
            1. Find test error for all different hypothesis, and pick the one with minimum test error (not fair)
            2. The hypothesis gives better result only on given test set can not necessarily be reported as generalized results (generalized test error)
            3. This is similar to analogy of overfitting
            4. Approach is to split data in 3: Training (60%), cross-validation (20%). Test (20%) set
            5. We compare cross-validation error to select best model and then find generalized test error by applying model on the test data set
    c. Bias vs Variance problem:
        i. As we increase degree of polynomial, the training error decreases
        ii. Whereas, cross-validation/ test error would take a U-shaped curve
        iii. For high bias (underfit) problem, train and test error will be high
        iv. For high variance (overfit) problem, train will be low but test will be high



    d. Regularization and bias/ variance:
        i. When lambda is very large → thetas are heavily penalized → underfit
        ii. When lambda is very small → very small penalty → overfit

iii. Process:
   1. Make multiple models for multiple value of lambdas
   2. For each case find thetas corresponding to minimum cost J
   3. Use these thetas to find cross validation error for each model
   4. And then pick (and report test error) the lambda where cross validation is minimum



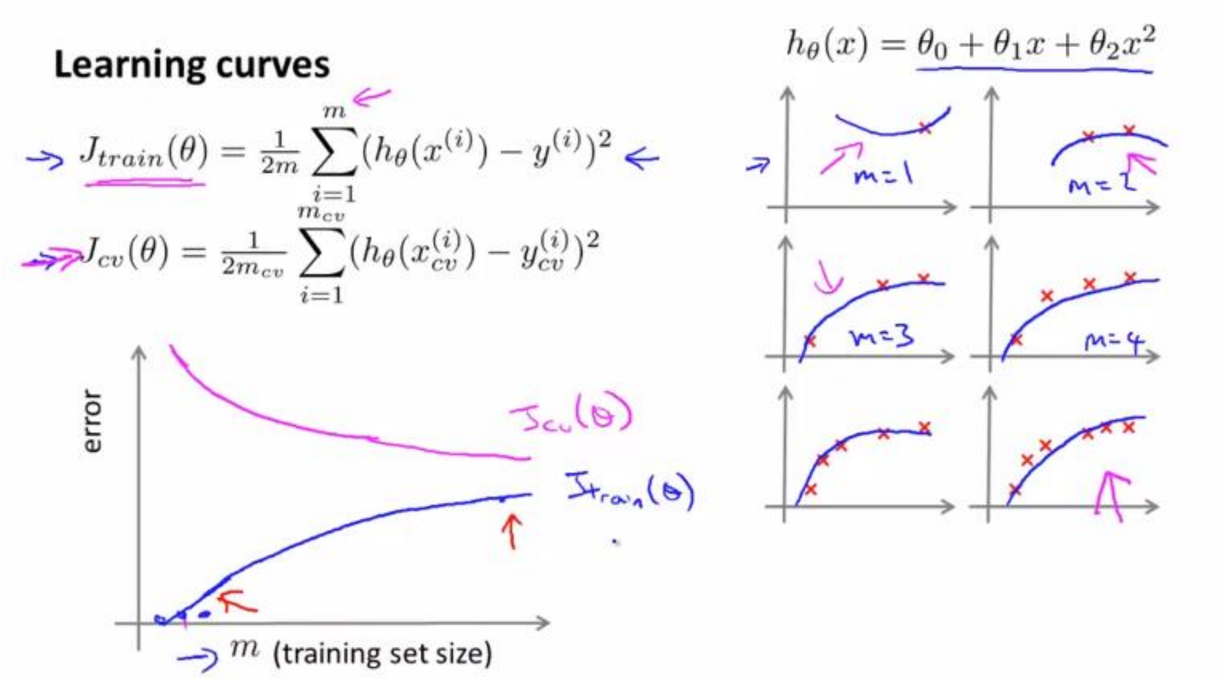
Bias/variance as a function of the regularization parameter λ

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

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

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

e. Learning Curves:
   i. Plot train and cross-validation over size of training set



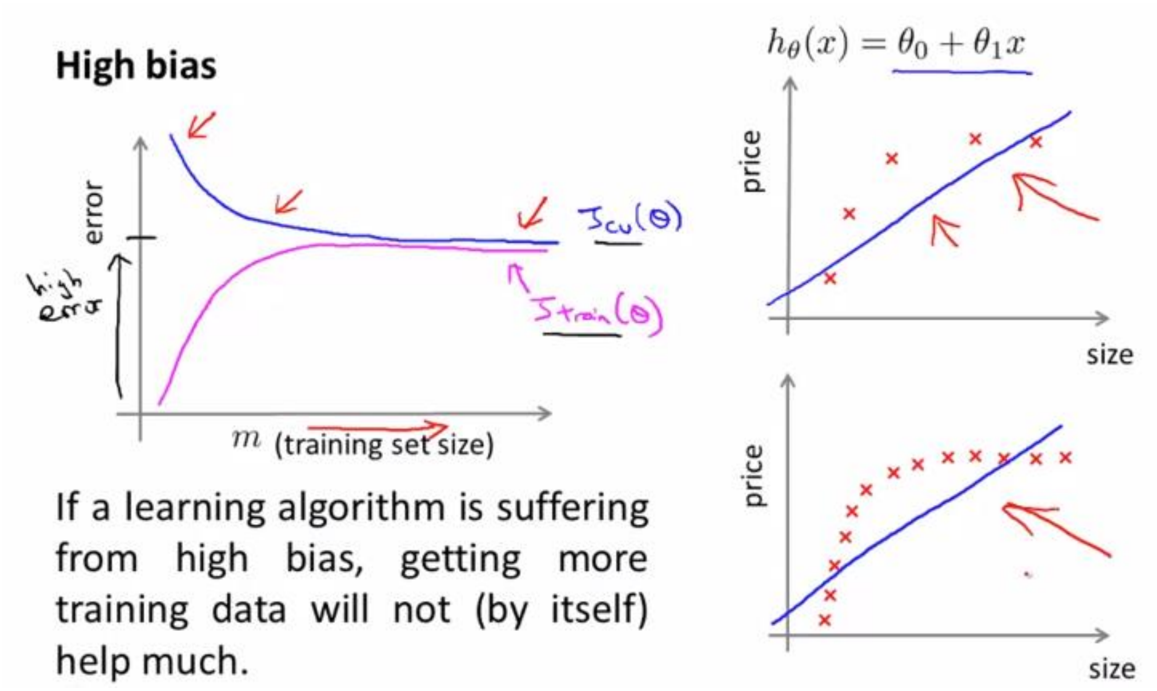
Learning curves

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

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

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

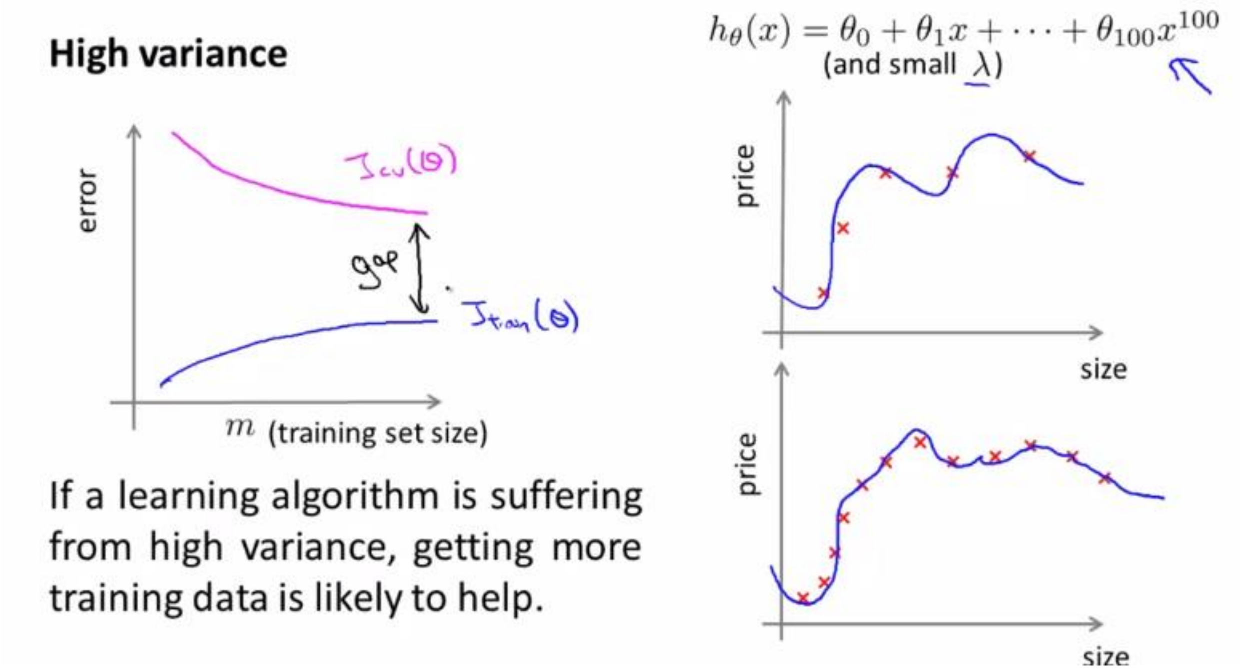
   ii. In high bias model (underfit model), getting more training data will not help much. Indication is that train and cross-validation are fairly similar.



High bias

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

iii. In high variance case (overfit model), getting more data will help because eventually the cross-validation will keep decreasing.



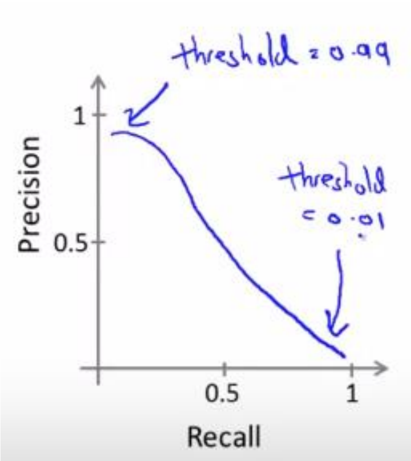
High variance

If a learning algorithm is suffering from high variance, getting more training data is likely to help.

3. Machine Learning System Design:
   a. Prioritizing (for a spam filter classifier):
      i. Pick n (10k to 50k) most frequently occurring words in the training set as spam/ not spam indicative word (represent feature matrix as a bag of word model from this list)
   b. Error Analysis:
      i. Approach:
         1. Make simple algo quickly and test it on cross-validation data
         2. Plot learning curves to decide if need more data, features, etc (can't be determined in advanced)
         3. Error Analysis: manually examine error examples. Spot systemic trend in these errors
   c. Error Metrics for Skewed Classes (for cancer diagnosis problem):
      i. When ratio of positive and negative class is extreme, it is skewed classes problem, thus classification accuracy in these situations is not an ideal performance metric
      ii. Metrics like recall and precision needs to be calculated in these situations:

        1. Precision: TP/ (Predicted as positive) = TP/ (TP + FP)
        2. Recall: TP/ (Actual positives) = TP/ (TP + FN)
    iii. Note: y = 1 is given to rarer class for precision and recall calculation

d. Trading off precision and recall:
    i. Higher Precision, Low Recall:
        1. Change threshold of hypothesis from 50% to 70%. It means if the classifier is more than 70% accurate then only classify as cancer (because of sensitivity of cancer news for patient).
        2. It means higher fraction of people classified as having cancer will actually have cancer
    ii. High Recall, Low Precision:
        1. Avoid FN (avoid failing to diagnose cancer if they actually have it)
        2. Threshold can be changed from 50% to 30%. It means that even if classifier is 30% confident, it is better to classify as cancer (so we don't miss out on FN)
        3. It means we will be correctly flagging higher fraction of people who actually have cancer

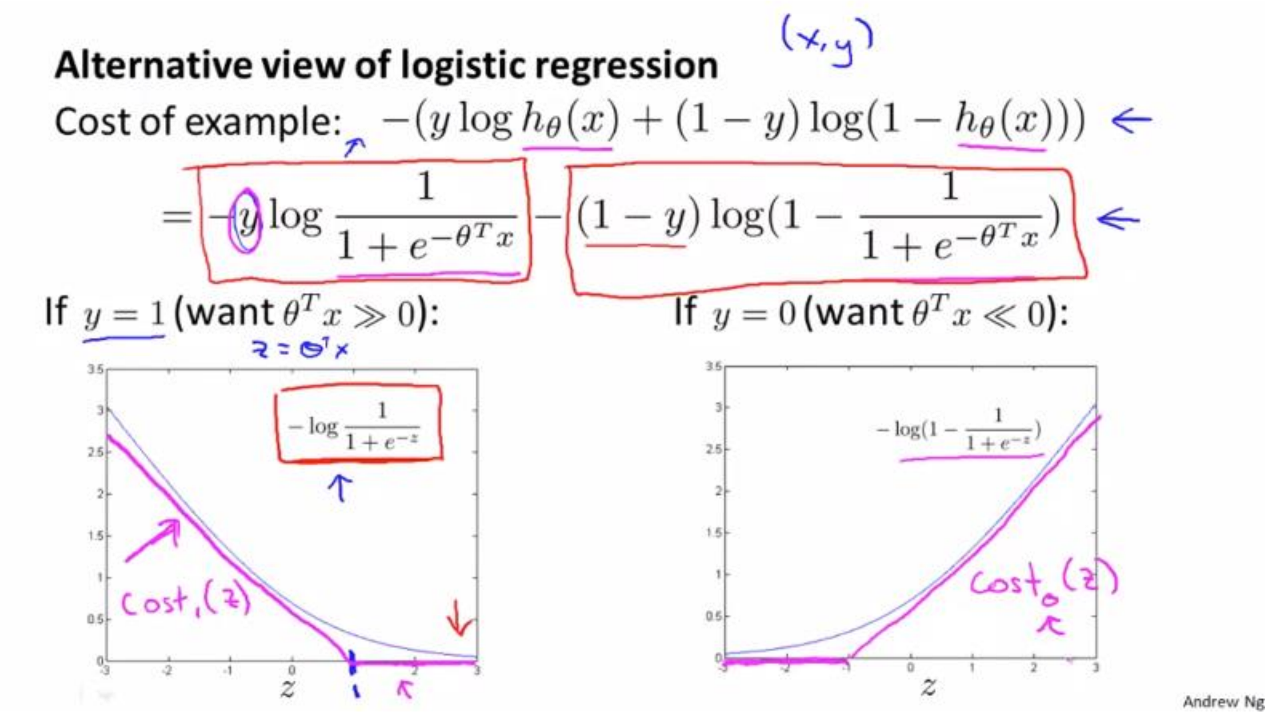


    iii. Precision – Recall curve can be of different shape depending on nature of classifier
    iv. How to compare different precision/ recall numbers (based on different threshold)?
        1. Now we have 2 numbers giving performance of the classifier
        2. We use F1 score – that is a single real number evaluation
        3. F1 Score = 2 * (P X R)/ (P + R)
        4. When P or R is 0, F score is 0. Only when both are 1, F score is 1.

e. Data for ML:
    i. Usually algos give better performance when feed more data
    ii. Try imagining if an expert human in a field can confidently predict for our problem, if no, get more features/ knowledge on problem
    iii. Getting more data can be useful when we use low bias algorithms (NN with hidden) when train loss is low and cross-validation loss is comparable to training loss

Week 7:

1. Optimization objective: Support Vector Machine (SVM):
    a. Modification in Logistic regression cost for SVM



Alternative view of logistic regression (x,y)

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

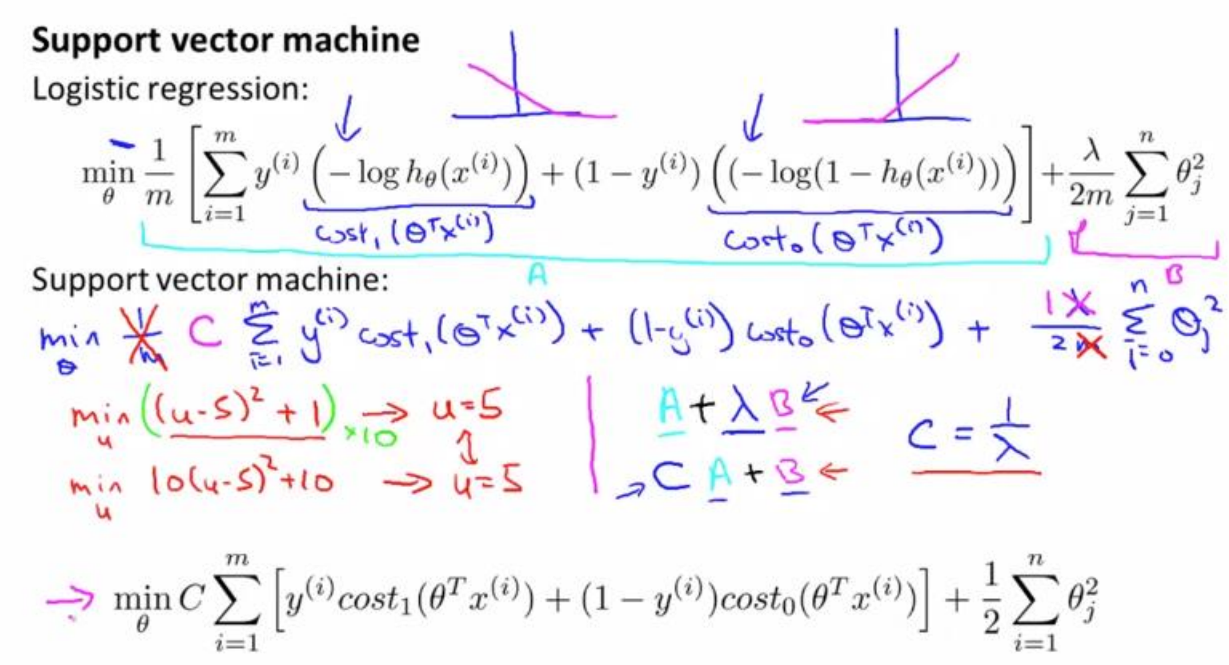
$$= -y\log\frac{1}{1+e^{-\theta^Tx}} - (1-y)\log\left(1-\frac{1}{1+e^{-\theta^Tx}}\right)$$

If $y=1$ (want $\theta^Tx \gg 0$): $z=\theta^Tx$

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

Andrew Ng

    b. Similarity in cost function of Logistic regression and SVM



Support vector machine

Logistic regression:

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

Support vector machine:

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

$\min_u \left((u-5)^2+1\right) \rightarrow u=5$ $\times 10$

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

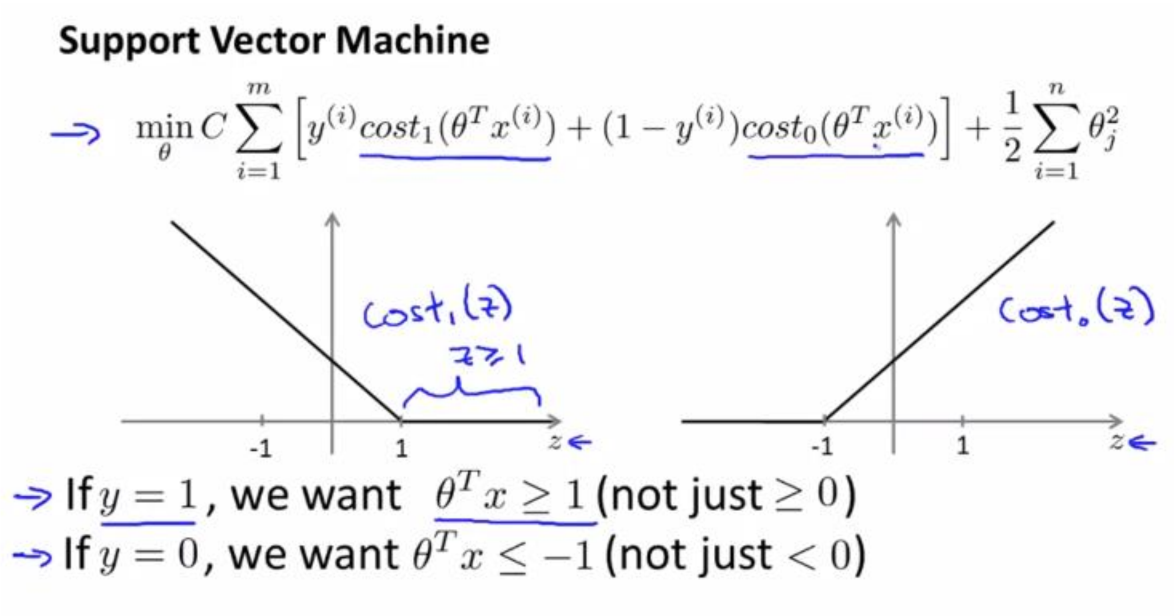
$A + \lambda B$

$C A + B$

$C = \frac{1}{\lambda}$

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

2. Large margin intuition: SVM
    a. Margin of hypothesis changes as shown:



Support Vector Machine

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

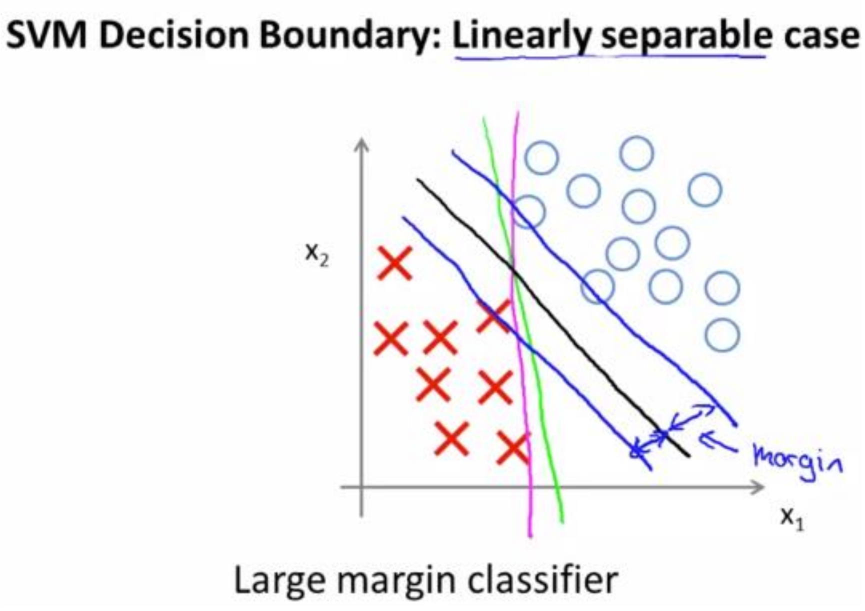
→ If $\underline{y=1}$, we want $\underline{\theta^Tx\geq 1}$ (not just ≥ 0)

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

b. The SVM equation can be written as follows: C.A + B

c. When C is very large constant, the A tern us forced to be 0

d. SVM: Choses a decision boundary that has a larger distance from minimum training examples of each class. This distance is called margin of SVM.



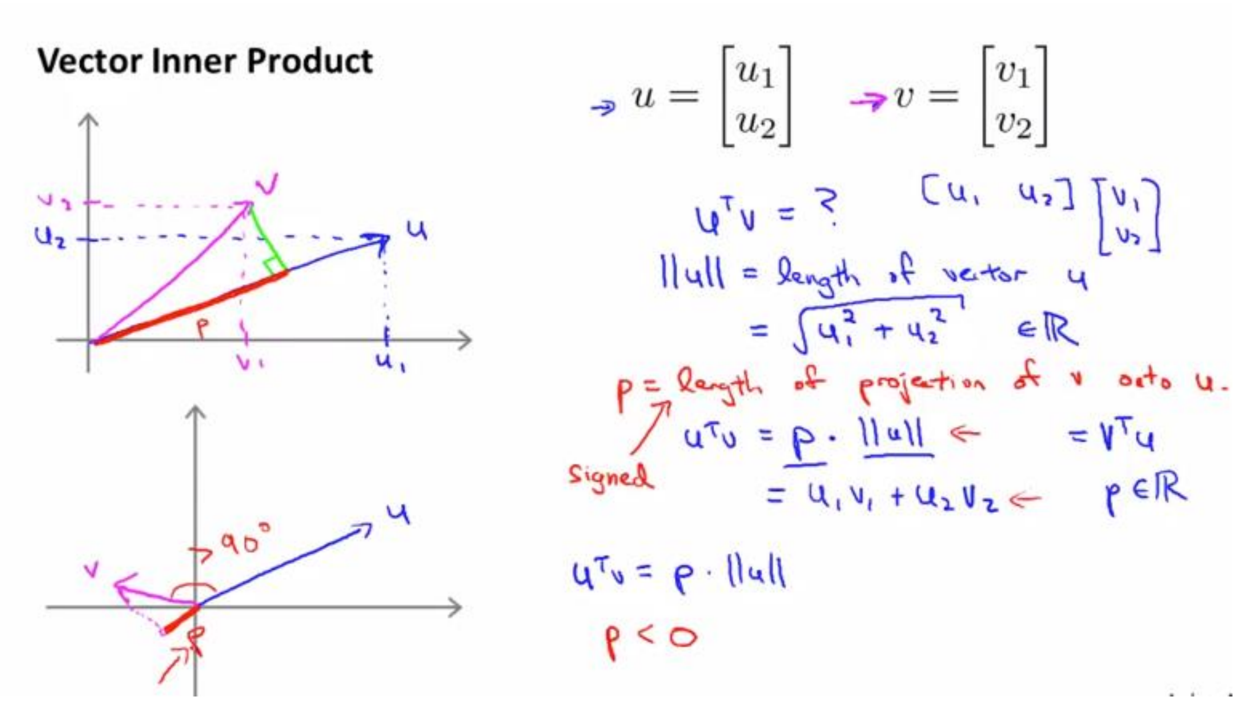
SVM Decision Boundary: Linearly separable case

Large margin classifier

e. A purely large margin classifier (only C is very large) can be prone to be outliers but SVM is much sophisticated as C can be varied to overlook these outliers.

3. Math for large margin classifiers

   a. Inner product of 2 vectors U and V is U' V



Vector Inner Product

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

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

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

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

p = length of projection of v onto u.

Signed

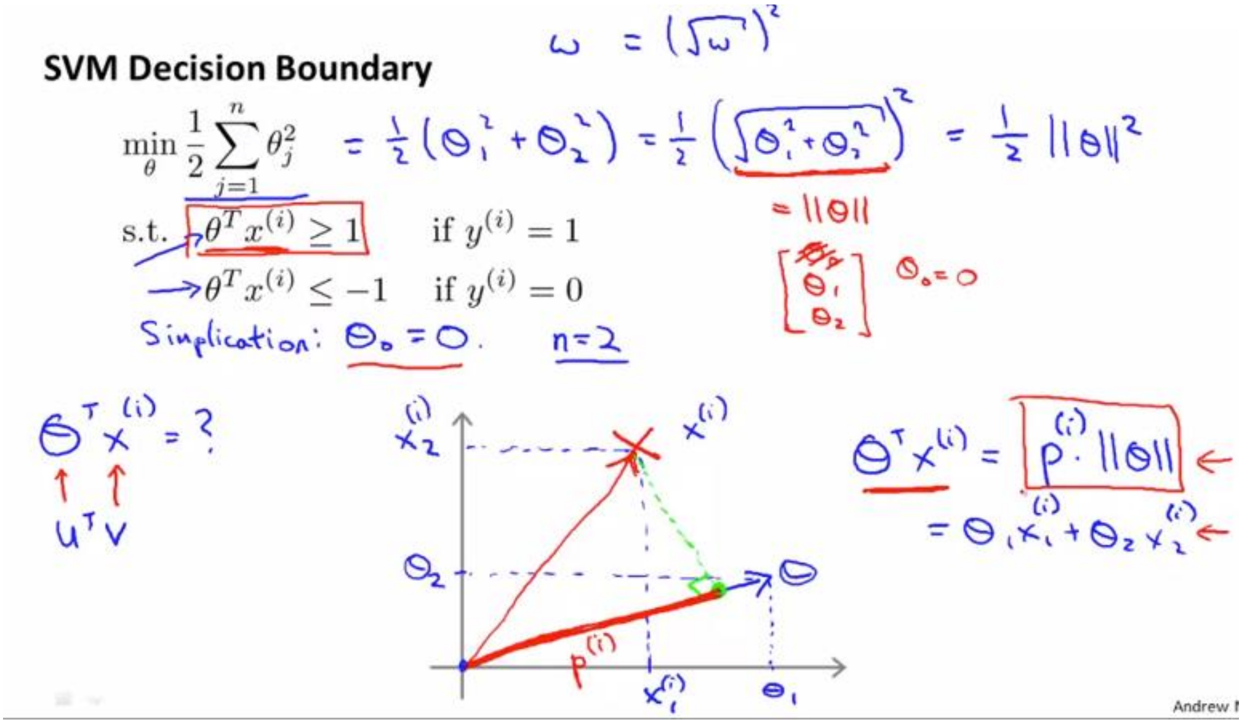
$u^T v = p \cdot \|u\| \leftarrow$ $= v^T u$

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

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

$p < 0$

b. SVM Decision boundary math



SVM Decision Boundary

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

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

$= \|\theta\|$

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

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

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

Simplication: $\theta_0 = 0$. $n = 2$

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

$\uparrow \uparrow$

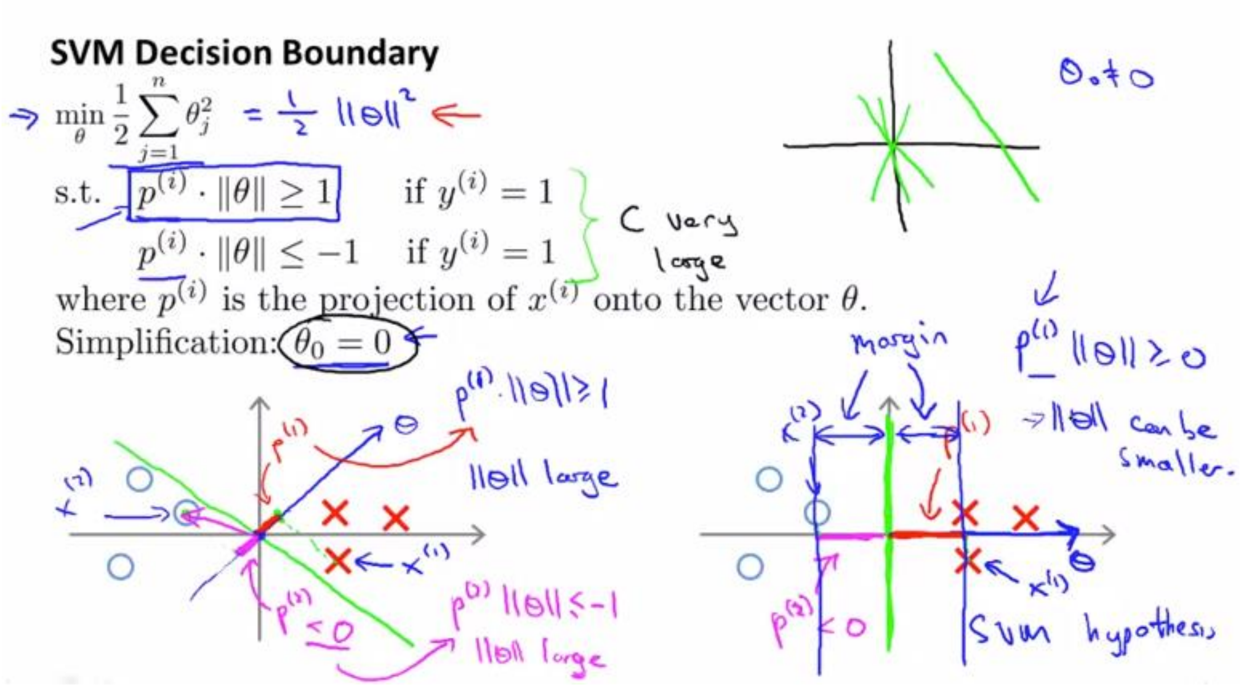
$u^T v$

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

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

c. Theta vector is orthogonal to the decision boundary for SVM

d. Hypothesis corresponding to decision boundary where projection vector is large is chosen because that will allow theta is be smaller which is ultimately our goal



SVM Decision Boundary

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

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

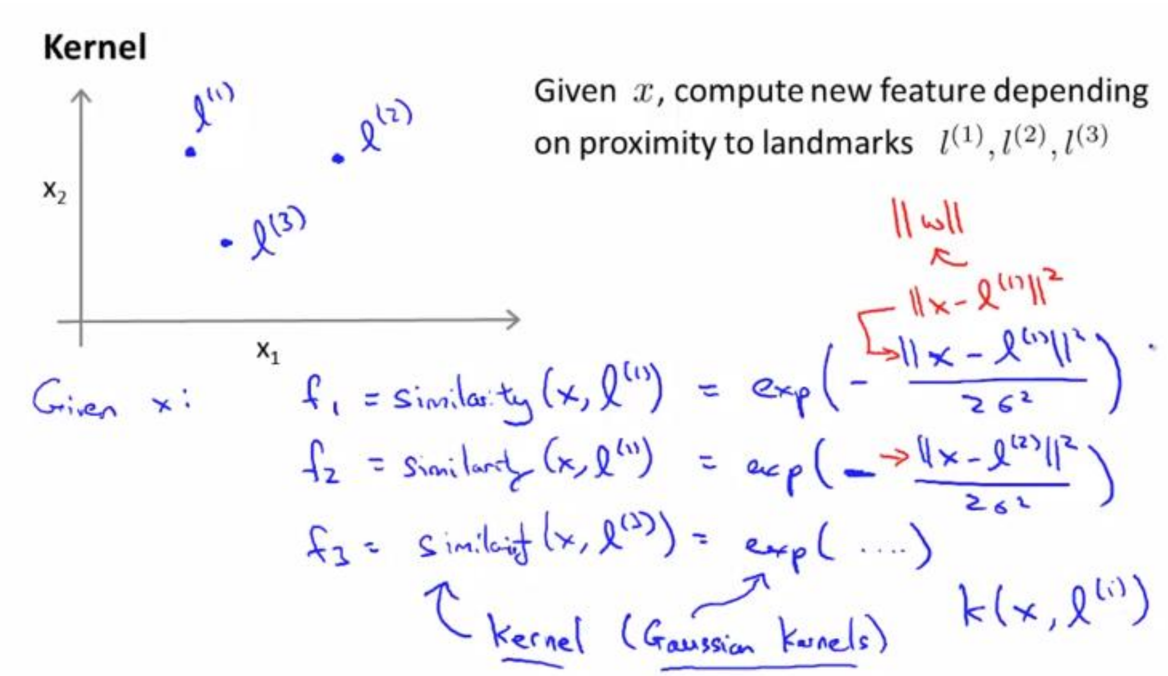
$p^{(i)}\cdot\|\theta\| \leq -1$ if $y^{(i)} = 1$

C very large

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

Simplification: $\theta_0 = 0$

4. Kernel 1:
    a. The similarity function (feature) used is called Kernel



Kernel

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

Given x:

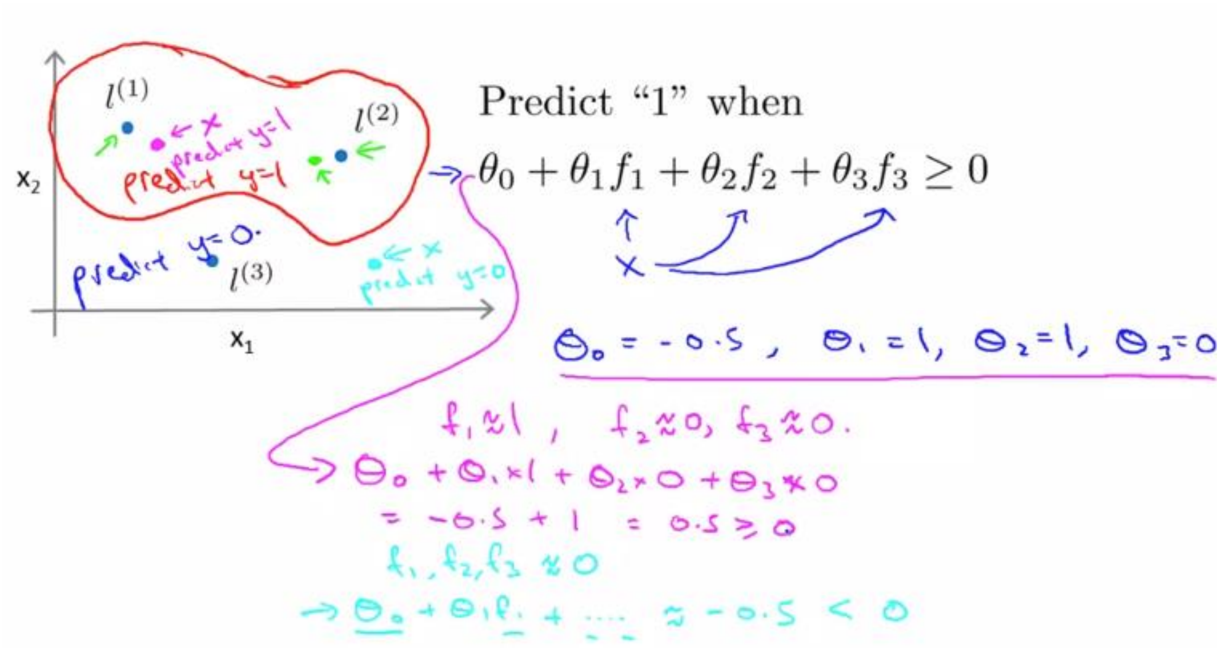
$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

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

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

kernel (Gaussian kernels) $k(x, l^{(i)})$

    b. When x is very close to landmark, then corresponding f/ similarity/ kernel value would be 1
    c. When x is very far from landmark, then corresponding f/ similarity/ kernel value would be 0
    d. Sigma term will determine how rapidly the f/ similarity function will degrade when the data is far from landmark
    e. Complex non linear boundaries can be developed by carefully selecting landmark, and thetas

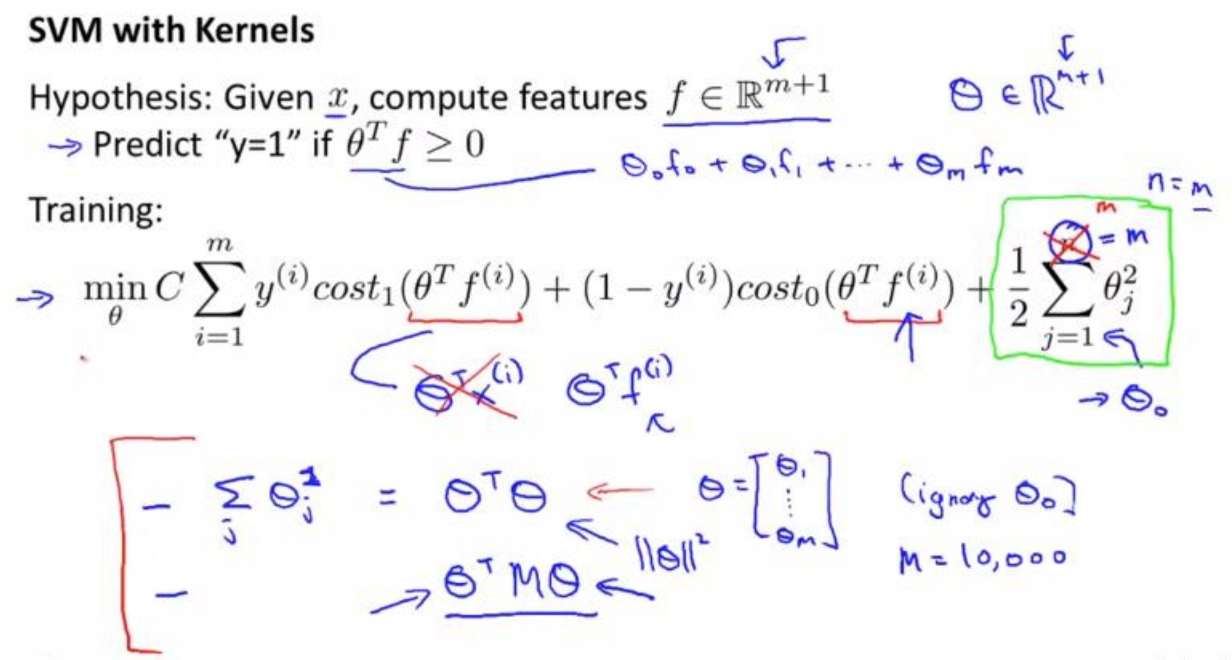


$l^{(1)}$
$l^{(2)}$
$l^{(3)}$
x_1
x_2
predict y=1
predict y=1
predict y=0.
predict y=0
$\Theta_0 = -0.5, \Theta_1 = 1, \Theta_2 = 1, \Theta_3 = 0$
$f_1 \approx 1, f_2 \approx 0, f_3 \approx 0.$
$\Theta_0 + \Theta_1 \times 1 + \Theta_2 \times 0 + \Theta_3 \times 0$
$= -0.5 + 1 = 0.5 \geq 0$
$f_1, f_2, f_3 \approx 0$
$\Theta_0 + \Theta_1 f_1 + \dots \approx -0.5 < 0$

Predict "1" when

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

5. Kernel 2:
    a. How to get landmarks?
    b. Landmarks would be exactly same points as training examples. Eventually we will have m landmarks if we have m training examples
    c. Using all training examples, we establish landmarks. Using landmarks, we find kernels/ f/ similarity and accumulate them to build a feature vector
    d. The second term is slightly modified as [theta' * M * theta] because the M term will help in scaling for larger problems
    e. We can apply the kernel idea to linear regression but then it will be computationally very expensive



SVM with Kernels

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

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

Training:

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

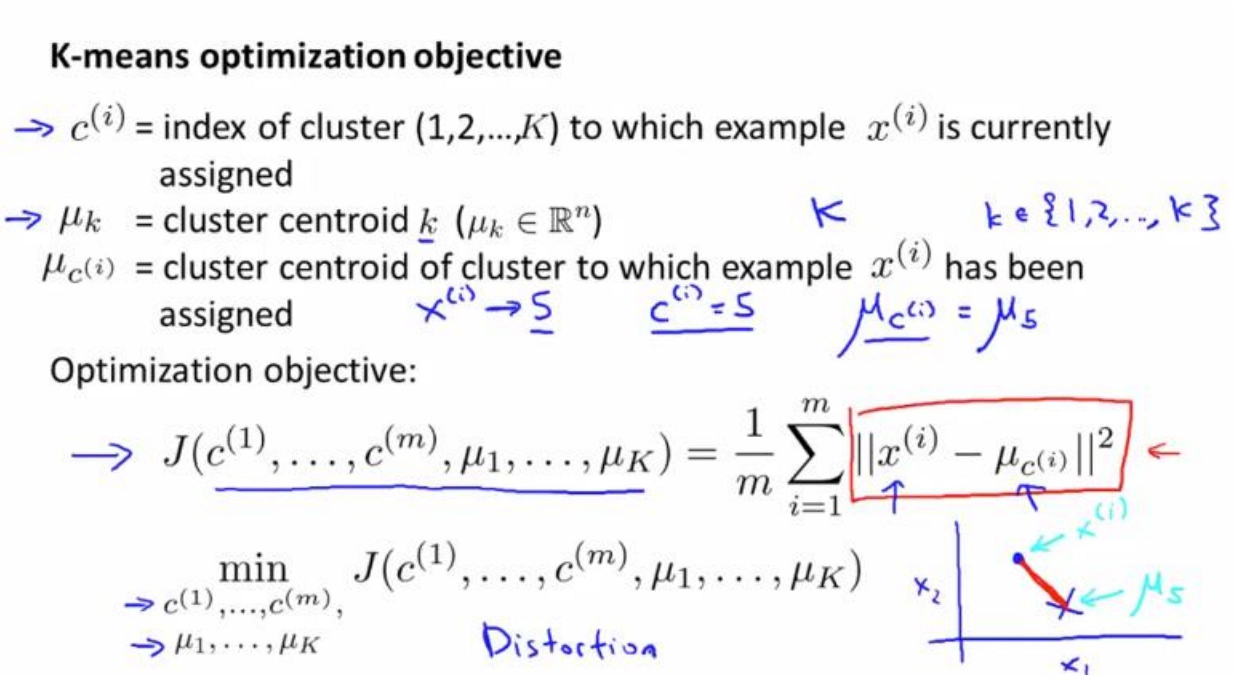
$\Theta \in \mathbb{R}^{m+1}$
$\Theta_0 f_0 + \Theta_1 f_1 + \dots + \Theta_m f_m$
$n = m$
$= m$
$\cancel{\Theta^T x^{(i)}}$
$\Theta^T f^{(i)}$
$\to \Theta_0$
$\sum_j \Theta_j^2 = \Theta^T \Theta$
$\Theta = \begin{bmatrix} \Theta_1 \\ \vdots \\ \Theta_m \end{bmatrix}$
(ignore Θ_0)
$\|\Theta\|^2$
$\Theta^T M \Theta$
$M = 10,000$

    f. Choose value of C (= 1/ lambda)
        i. Large C: low bias, high variance (more prone to overfitting)
        ii. Low C: high bias, low variance (more prone to underfitting)
    g. Choose sigma square term (gaussian term)
        i. High sigma square: smooth and varies smoothly (high bias, low variance)
        ii. Low sigma square: less smooth (low bias, high variance)
6. Using SVM:
    a. Software libraries for SVM – liblinear, libsvm

b. Our task during implementation:
    i. Choice of C
    ii. Choice of Kernel (similarity)
        1. No kernel (linear) – n large, m small
        2. Gaussian – n small, m large (need to choose sigma square term)
        3. Need to satisfy Mercer's Theorem on any similarity function used. (avoid implementation problems)
        4. Other kernels:
            a. Polynomial Kernel ((X' * Theta + constant) ^degree of exp.)
            b. String kernel
            c. Chi-square kernel
            d. Histogram intersection kernel
c. Multi-class classification in SVM:
    i. Use inbuilt SVM libraries
    ii. Use one vs all method – build K different SVM models and get K different theta vectors
d. Logistic Regression vs SVM:
    i. Use logistic or SVM (linear kernel) when n is large (relative to m) (n=10000, m=10 to 1000)
    ii. Use SVM (Gaussian kernel) when n is small and m is intermediate (n=1 to 1000, m=10-10000)
    iii. Use SVM/ logistic with more features when n is small, m is large (n=1 to 1000, m=50000+)
    iv. Well designed NN can work for all above situations, but will be slower to train
e. Note: SVM's optimizer is convex optimization therefore global optimum will be reached

Week 8:

1. Unsupervised Learning:
    a. Given data has no labels associated and ask algo to find structure in data
    b. If the structure found is cluster, then they are clustering algorithms
    c. Applications of Clustering:
        i. Market segmentation
        ii. Social network analysis
        iii. Organize computing clusters
        iv. Astronomical data analysis
2. Clustering: K-means algorithm
    a. Iterative algorithm which does two things:
        i. Cluster assignment: goes through all data points and assigns cluster to them based on distance from cluster centroid
        ii. Move centroid: move centroid to the mean of new clusters
    b. Repeat until centroid no longer changes
    c. Inputs of K-means:
        i. Value of K (number of clusters)
        ii. Training set X
3. Clustering: K-means optimization objective
    a. Objective is to minimize the squared distance of training example to the clustering centre it has been assigned



K-means optimization objective

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

→ μ_k = cluster centroid $\underline{k}$ ($\mu_k \in \mathbb{R}^n$) K $k \in \{1,2,\ldots,K\}$

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned $x^{(i)} \rightarrow \underline{5}$ $\underline{c^{(i)} = 5}$ $\underline{\mu_{c^{(i)}}} = \mu_5$

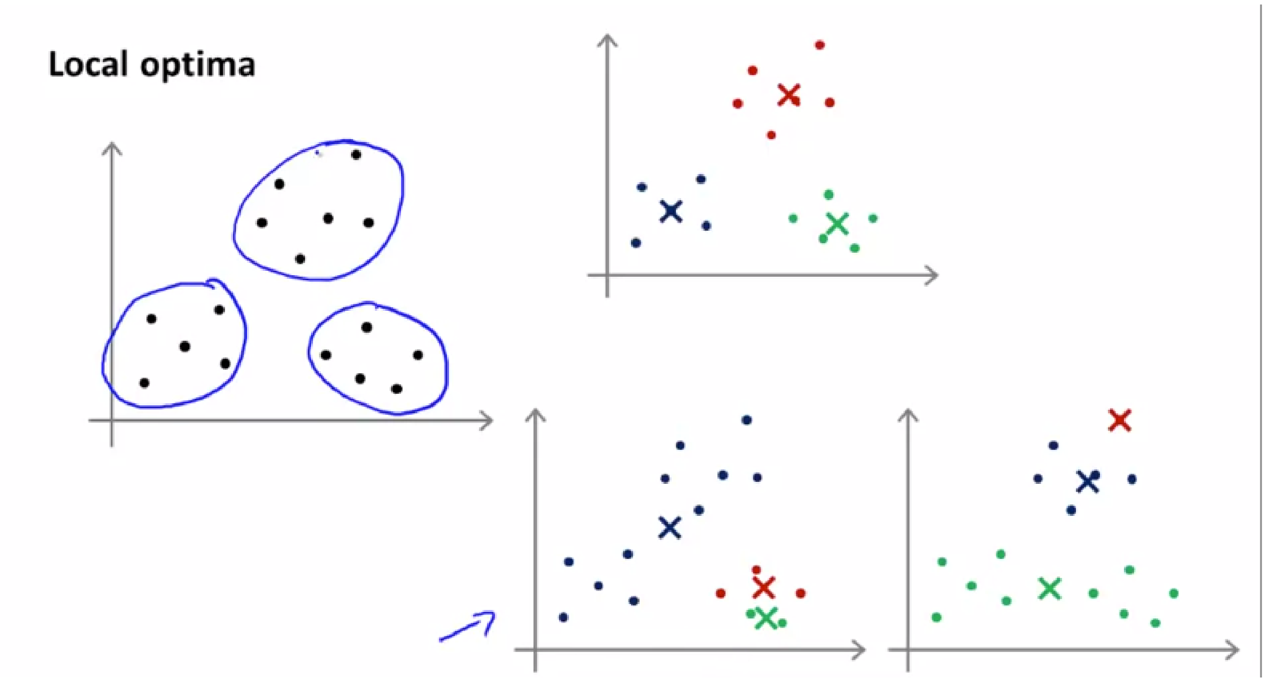
Optimization objective:

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

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

Distortion

    b. Clustering assignment step is minimizing J w.r.t clusters assigned to X keeping centroid fixed
    c. Move centroid step minimizes J w.r.t centroid keeping X fixed
4. Random initialization:
    a. Avoiding local optimum
    b. Steps to randomly initialize initial cluster centroids:
        i. Always have K < m
        ii. Randomly pick any K examples
        iii. Set K centroid equal to these examples
    c. K-means can converge to different clusters based on how initial centroids were defined



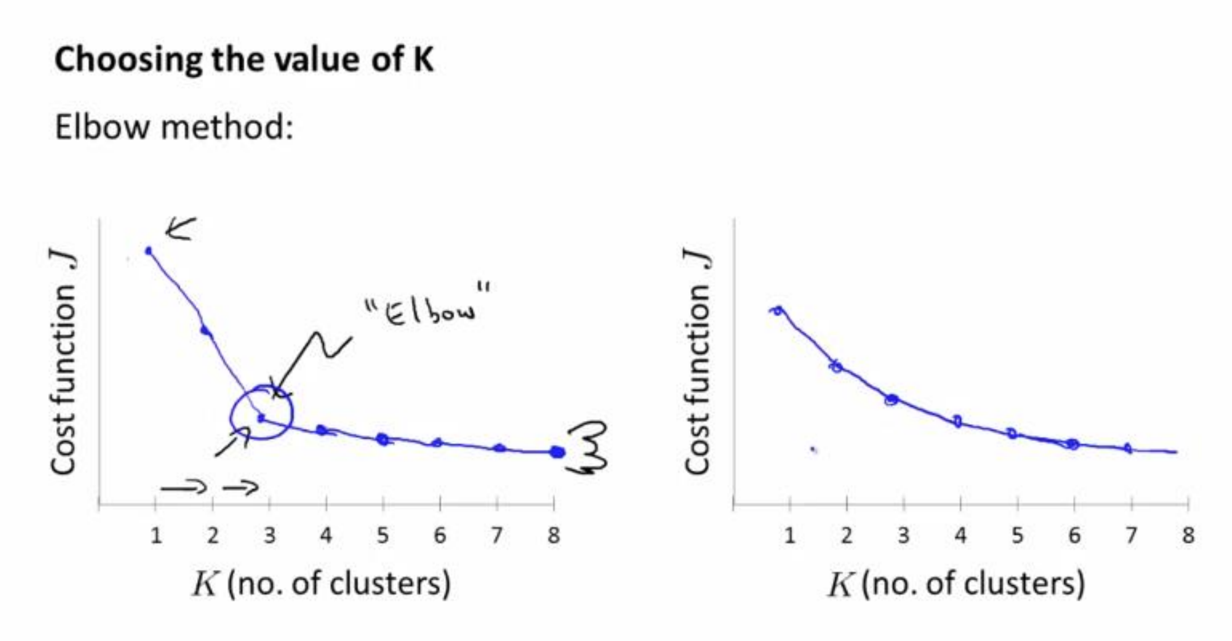
d. Due to random initialization, some local optima can be formed as shown above can be formed

e. To avoid this problem, K-means can be run multiple times (usually between 50 to 100) with random centroid initializations and then pick the one with minimum error/ cost/ distortion. (method will help when k is between 2 and 10)

5. Chose K:

   a. One common way is to look at it manually by looking at visualizations/ output of clustering algo. Etc.

   b. Elbow method:

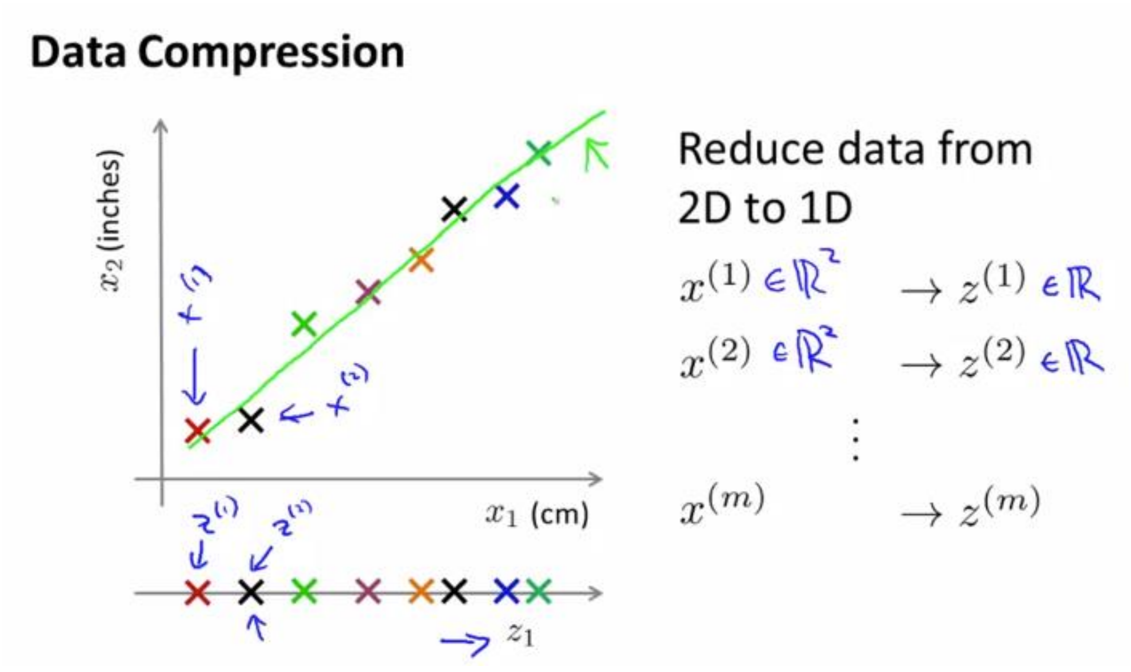


   c. How values of K serve to our ultimate problem

6. Dimensionality reduction: data compression (unsupervised learning problem)

   a. Why? Data Compression

   b. To reduce redundancy (like to reduce data from 2D to 1D)



Data Compression

Reduce data from 2D to 1D

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

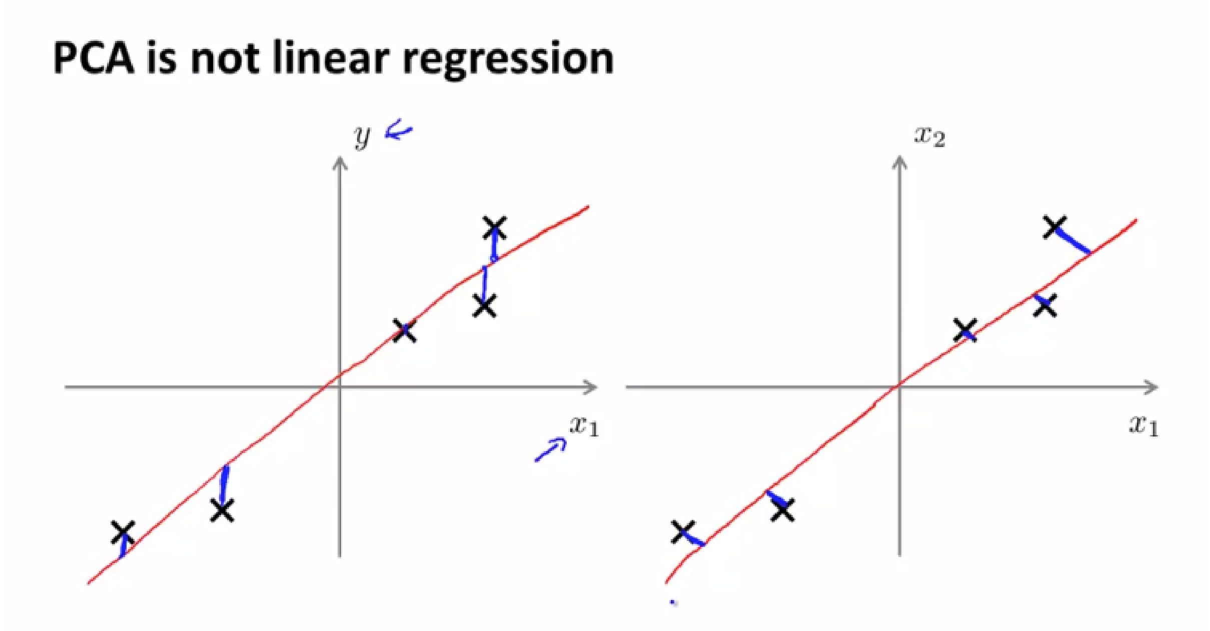
$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$

$\vdots$

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

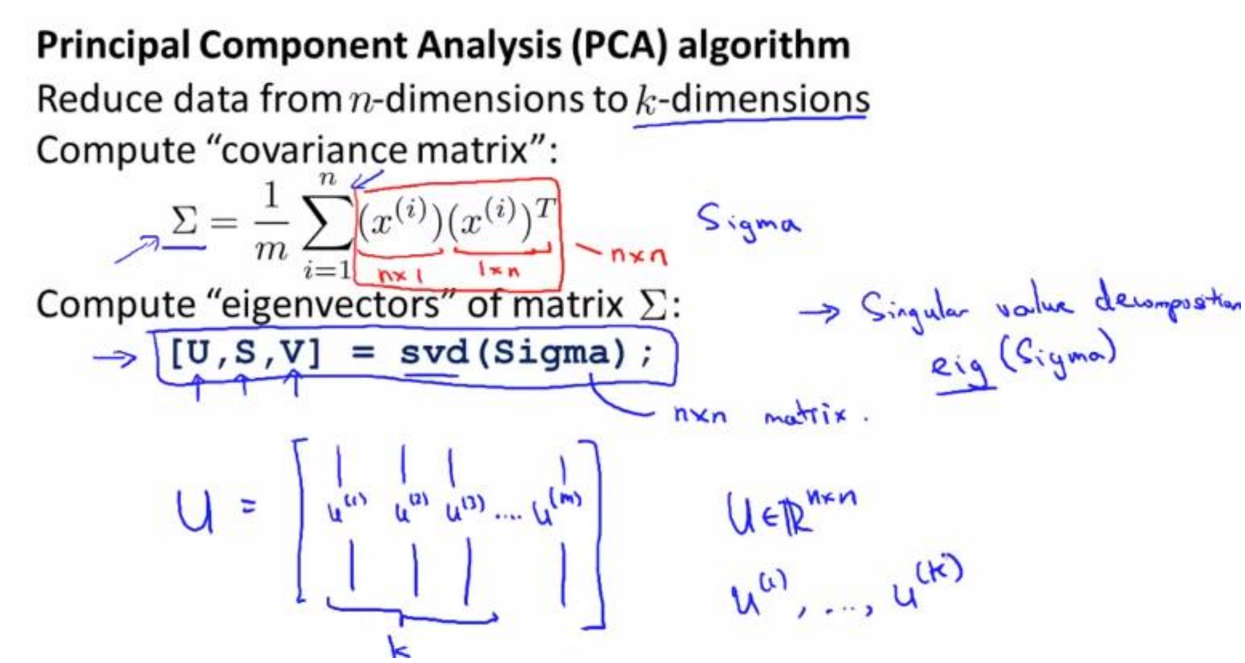
   c. Typically, dimensional reduction can reduce 1000 D data → 100 D data

7. Dimensionality reduction: data visualization
   a. The reduced new dimensions don't have any meaning, it is up to us to infer meaning out of them
8. Principal Component Analysis (PCA): Problem formulation
   a. PCA finds a lower dimensional surface onto which data is projected so as to minimize the sum of square of the orthogonal distance between the actual data point and its projection on the surface (minimize projection squared error)
   b. Also, it is advised to perform feature scaling and mean normalization before applying PCA
   c. Reducing n dimension data to k dimension by finding k vectors onto which the data is to be projected
   d. PCA seems like linear regression but it is not linear regression at all
      i. In PCA, x1,x2, x3…. Are different dimensions (and are treated equally)
      ii. In linear regression, we find relationship between output y and x



PCA is not linear regression

9. Principal Component Analysis (PCA): Algorithm
   a. First, we need to find U vector (corresponding to k new direction vectors) and we need z (from z1 to zk which is a dimensionally reduced version of X which was from X1 to Xn)
   b. To reduce n dimension to k dimension, we compute covariance (sigma) matrix which is (n x n) as:



Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

Sigma

Compute "eigenvectors" of matrix Σ:

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

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

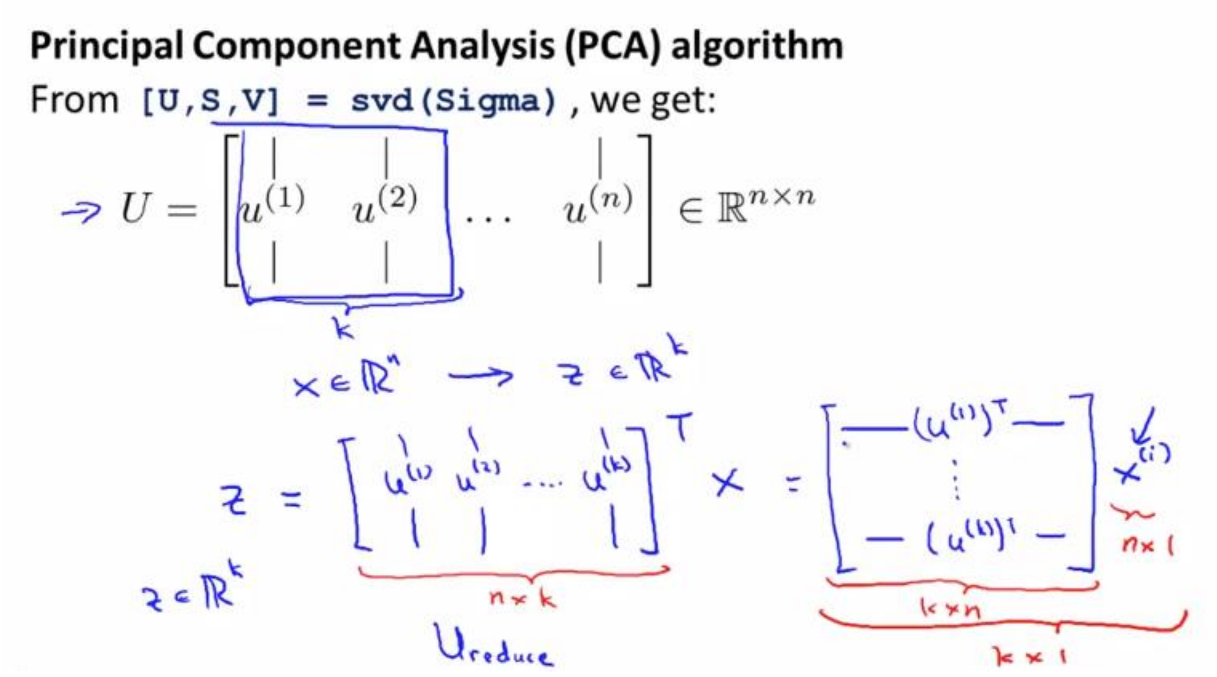
$\underbrace{u^{(1)}\ u^{(2)}\ u^{(3)}}_{k}$

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

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

c. Then we find eigenvectors of matrix sigma (covariance)

d. Output of svd in octave/ matlab are U, S, V matrices out of which we take the first k columns of k as new dimensions and to find Z (reduced data), we do the following computation:



Principal Component Analysis (PCA) algorithm

From `[U,S,V] = svd(Sigma)`, we get:

$$U = \begin{bmatrix} \boxed{\begin{matrix} | & | \\ u^{(1)} & u^{(2)} \\ | & | \end{matrix}} & \dots & \begin{matrix} | \\ u^{(n)} \\ | \end{matrix} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

k

$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$

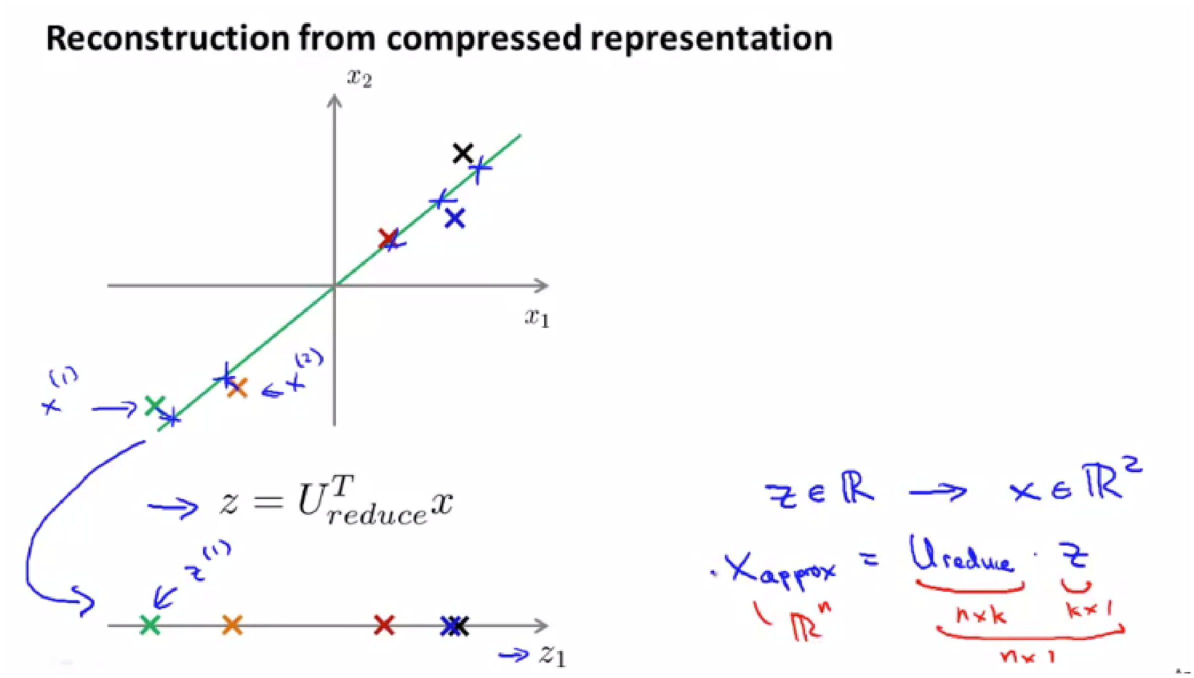
$$z = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}}_{n\times k}^{T} x = \underbrace{\underbrace{\begin{bmatrix} — (u^{(1)})^T — \\ \vdots \\ — (u^{(k)})^T — \end{bmatrix}}_{k\times n} \underbrace{x^{(i)}}_{n\times 1}}_{k\times 1}$$

U_{reduce}

$z \in \mathbb{R}^k$

10. Reconstruction from compressed representation:

a. Going from compressed dimension back to original



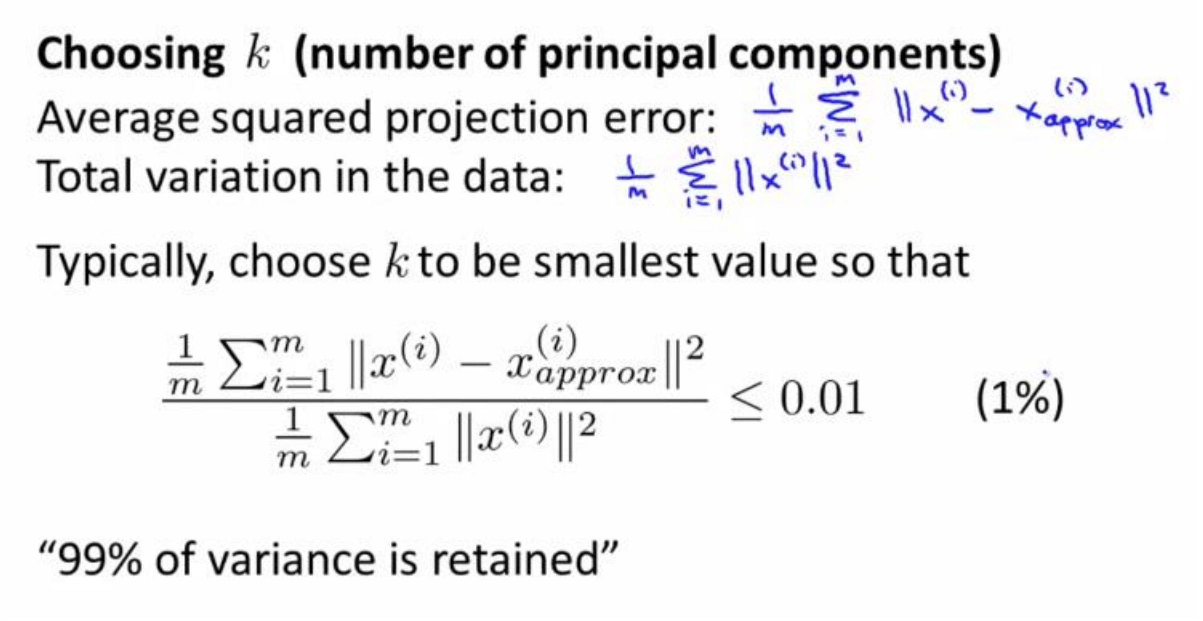
Reconstruction from compressed representation

$$z = U_{reduce}^T x$$

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

$$\underset{\in \mathbb{R}^n}{X_{approx}} = \underbrace{\underbrace{U_{reduce}}_{n \times k} \cdot \underbrace{z}_{k \times 1}}_{n \times 1}$$

11. Choosing K (number of principal components) in PCA:
    a. Ideally chose K such that, the ratio of average squared projection to the total variation in data (variance) is less than 1%



Choosing k (number of principal components)

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

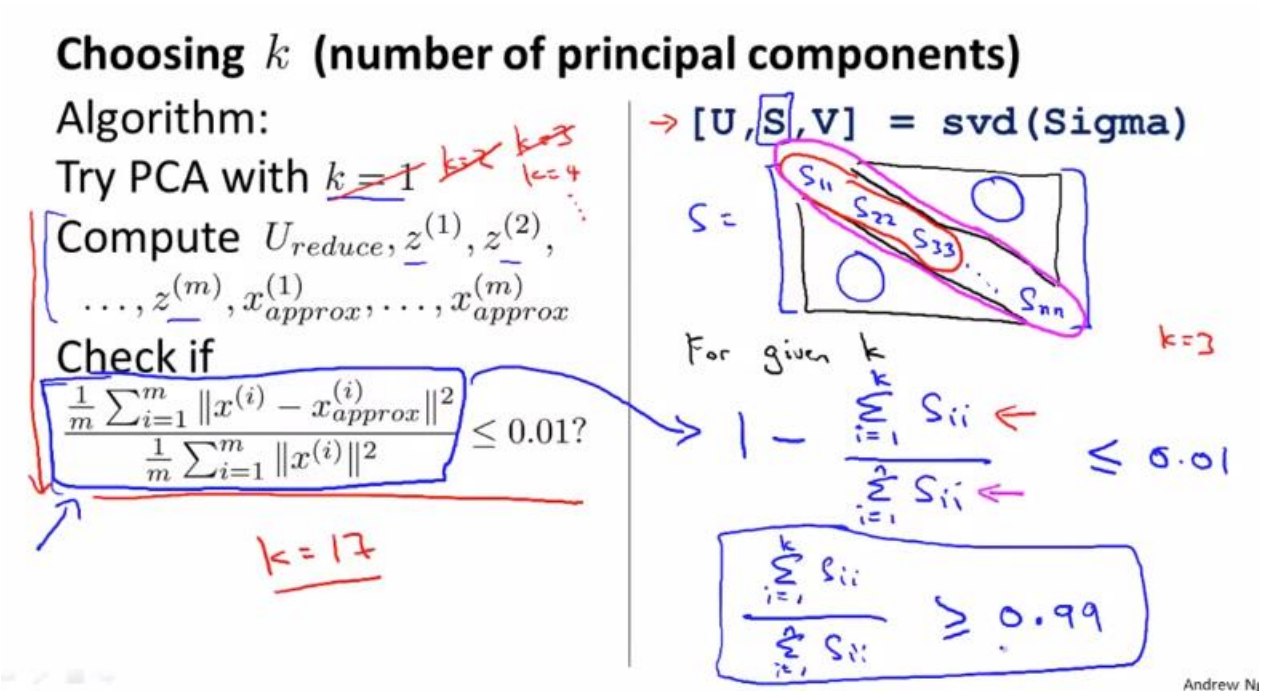
Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2$

Typically, choose k to be smallest value so that

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

"99% of variance is retained"

    b. One way to chose K is an iterative method and checking the ratio every time if it is less than 1% or not
    c. In octave/ matlab, the svd function returns U, S, V. Here, S is a diagonal matrix of size (n x n). The percentage term can easily be calculated from the S matrix as follows. Using this method we do not need to call the svd function iteratively and therefore is computationally less expensive, we only need to iterate over the S matrix:



Choosing k (number of principal components)

Algorithm:

Try PCA with ~~$k = 1$~~

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x^{(1)}_{approx}, \ldots, x^{(m)}_{approx}$

Check if

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \le 0.01?$$

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

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

For given k

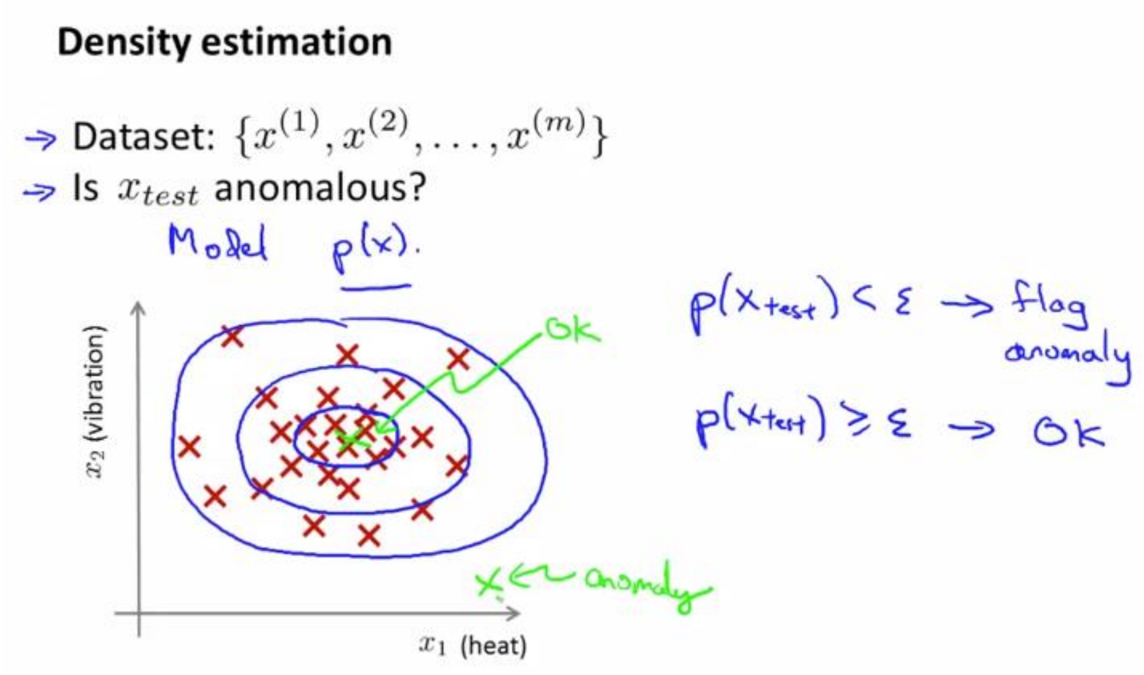
$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \le 0.01$$

$$\boxed{\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \ge 0.99}$$

12. Applied PCA:
    a. Supervised problem with data (X, Y) → Apply PCA to unlabelled X → New problem is with data (Z, Y)
    b. U reduced matrix is found only by running PCA on training set, later this U reduced is used as a mapping function when cross-validation and test set are executed
    c. Application of PCA:
        i. Compression: K is chosen to retain 99% of variance
            1. To reduce memory to store data
            2. Speed up the learning algorithm
        ii. Visualization: K is usually 2/ 3
    d. Misuse of PCA is to prevent overfitting because we have less features and hence less likely to overfit. The better way to address overfitting is to use regularization. This is because PCA while reducing dimension removes the information in the label Y of the dataset which methods like regularization do not do.

Week 9:

1. Anomaly Detection (E.g., quality control for aircraft engines):
    a. From unlabelled dataset, we will build a probability function p(x)
    b. Then while testing, we will compare p(x_test) to some threshold value, and if it is less than that, we will flag x_test as anomaly, otherwise it is normal



Density estimation

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

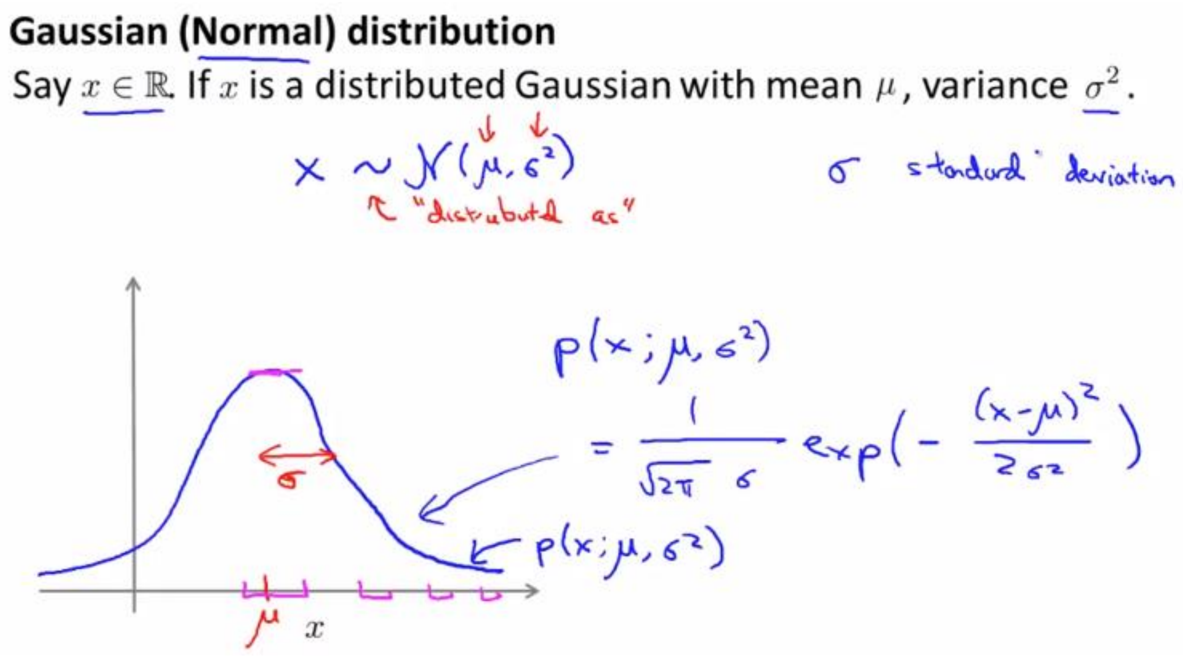
→ Is x_{test} anomalous?

Model $p(x)$.

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

$p(x_{test}) \geq \varepsilon$ → OK

    c. Similar model can be extended for Fraud detection taking user activities as features
    d. E.g., Computer manufacturing in data centre
2. Gaussian Distribution:
    a. It is shown as X ~ N (mu, sigma_square) [N is representation of Normal Distribution]
    b. Mu is the mean, and sigma represents 1 standard deviation



Gaussian (Normal) distribution

Say $x \in \mathbb{R}$. If x is a distributed Gaussian with mean μ, variance σ^2.

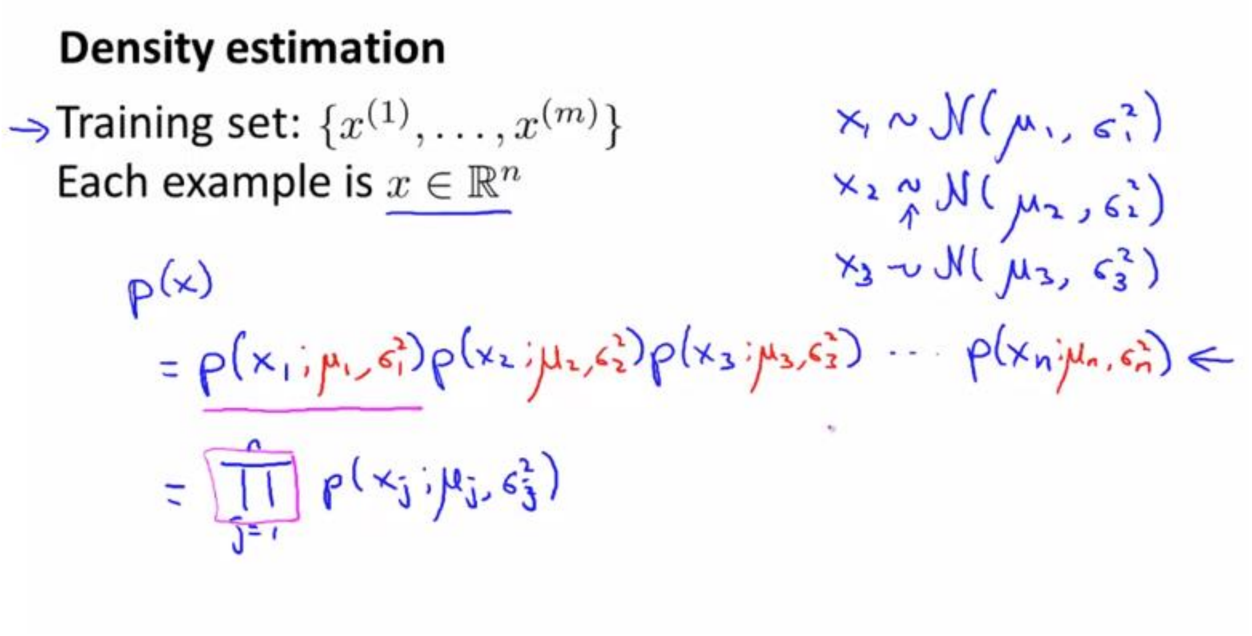
$x \sim \mathcal{N}(\mu, \sigma^2)$ ← "distributed as"

σ standard deviation

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

    c. Parameter Estimation:
        i. Given dataset which has gaussian distribution, we want to estimate mu and sigma_square
        ii. Formulas of mu and sigma_square are standard ones used in statistics

3. Anomaly detection Algorithm:
   a. We will model p(x) from the dataset
   b. All features are represented by unique gaussians with particular mean and sigma values
   c. Here, it is assumed that all features are independent but it also works fine otherwise



Density estimation

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

Each example is $\underline{x \in \mathbb{R}^n}$

$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$

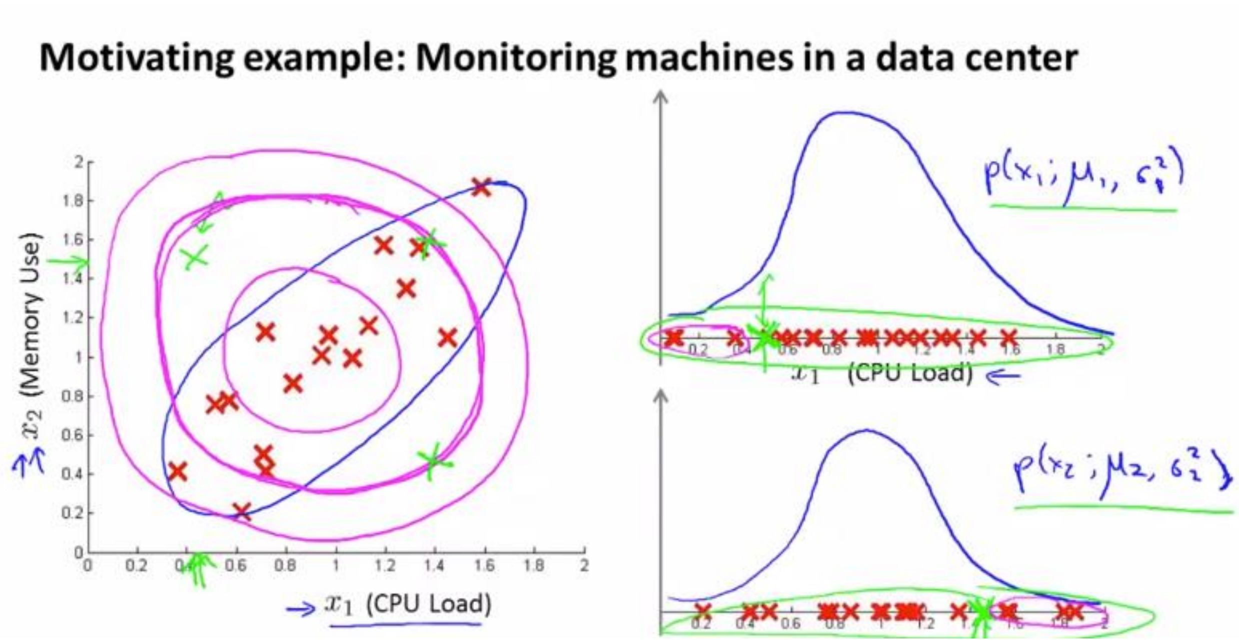
$x_2 \overset{\sim}{\uparrow} \mathcal{N}(\mu_2, \sigma_2^2)$

$x_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$

$$p(x) = \underline{p(x_1;\mu_1,\sigma_1^2)}\,p(x_2;\mu_2,\sigma_2^2)\,p(x_3;\mu_3,\sigma_3^2)\cdots p(x_n;\mu_n,\sigma_n^2) \leftarrow$$

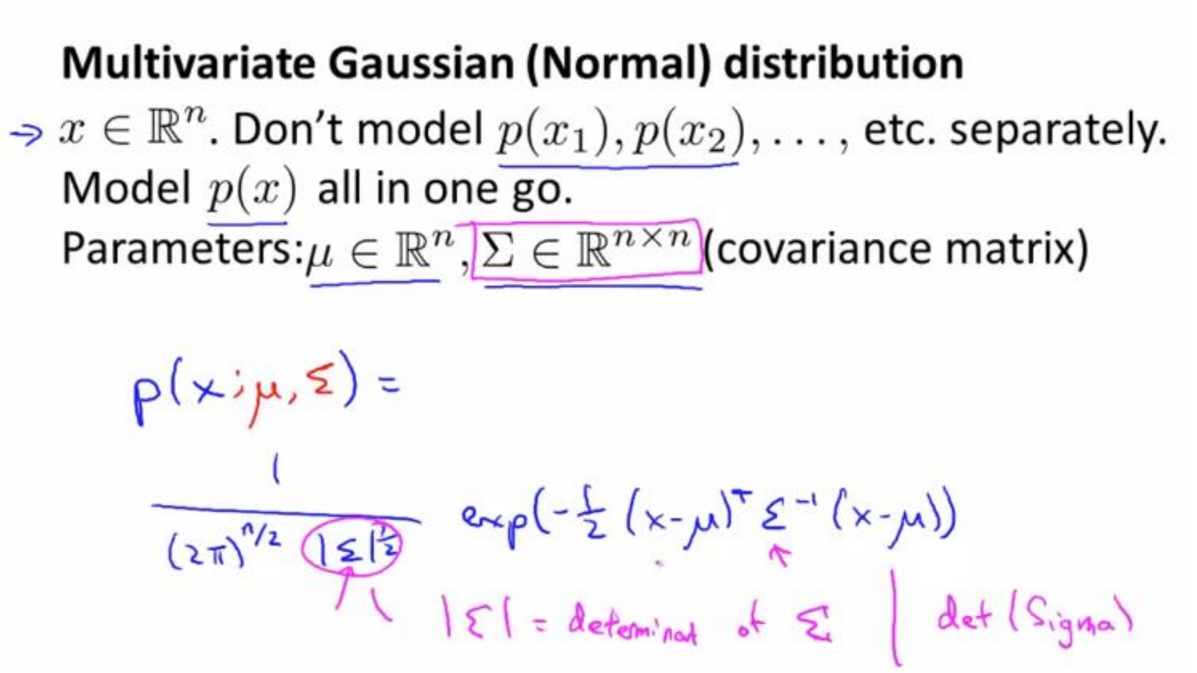
$$= \boxed{\prod_{j=1}^{n}} p(x_j;\mu_j,\sigma_j^2)$$

   d. Steps of algorithm:
      i. Chose x that are indicative of anomalous examples
      ii. From dataset, find gaussian parameters for each feature
      iii. Find p(x) (product of each gaussian probabilities) of new examples and then see if it is less than threshold or not
4. Developing and evaluating anomaly detection system:
   a. Real-number evaluation is very important for building ML models
   b. Assuming we have a labelled dataset, and treating it like a supervised learning problem
   c. Because of skewed dataset, we don't use accuracy as metric. We use confusion matrix, Precision/ Recall and F-1 Score
   d. We can try different values of threshold epsilon in cross-validation set and then pick the best one for testing dataset
5. Anomaly vs Supervised Learning:
   a. Small number of positive examples, Large negative examples → Anomaly detection
   b. Large number of positive examples, Large negative examples → Supervised
   c. Anomalies are of different types and future anomaly are nothing as seen before
   d. Positive and negative examples have enough dataset to get sense of both
   e. Fraud detection, manufacturing, and monitoring machines in data centre → Anomaly detection
   f. Email spam, Cancer classification, Weather prediction → Supervised
6. Feature selection for Anomaly detection:
   a. If features don't have Gaussian distribution, then apply some transformation to make it Gaussian
   b. Transformations like log (), power, exp, etc. can be used
   c. To pick features, we use error analysis. That means that p(x) must be large for normal and very small for anomaly examples
   d. Observe the data and then make a feature that grows very large or small in case of anomaly
7. Multivariate Gaussian Distribution:
   a. In the figure below, the normal gaussian fails



Motivating example: Monitoring machines in a data center

b. In multivariate gaussian, we will model variables together (using covariance)



Multivariate Gaussian (Normal) distribution

$x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2), \ldots,$ etc. separately.

Model $p(x)$ all in one go.

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

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

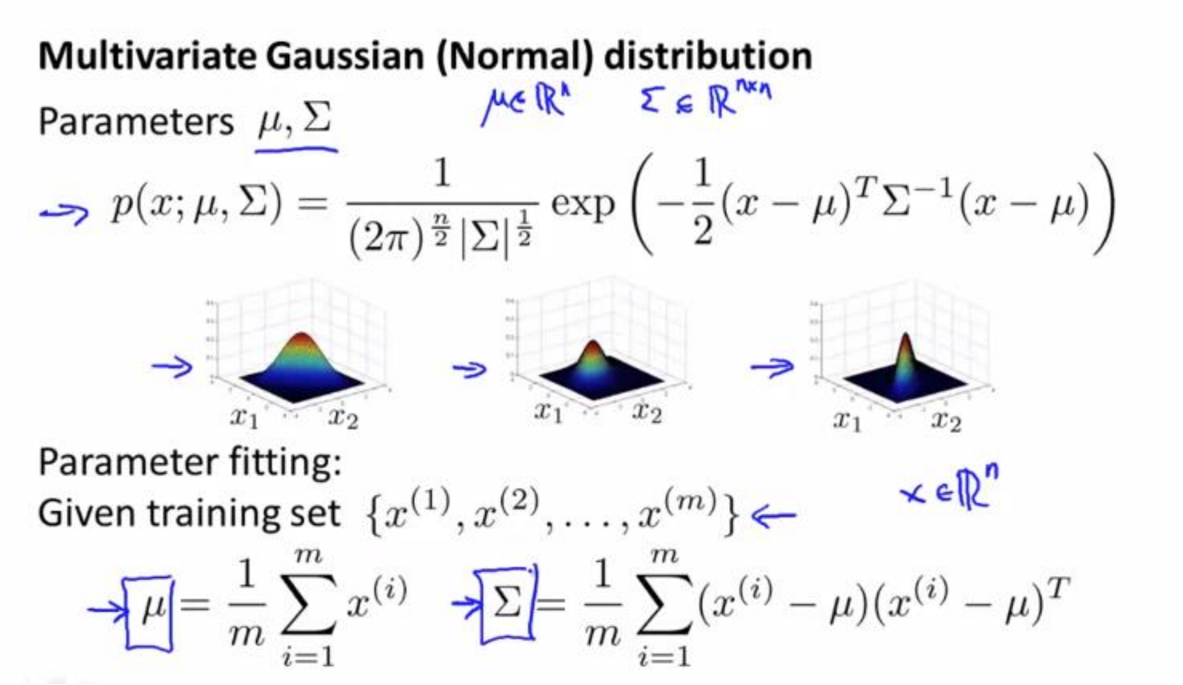
$|\Sigma|$ = determinant of Σ | det(Sigma)

c. Sigma is (each diagonal element corresponding to the respective variable in Gaussian), so when we increase it, the width of gaussian increases therefore it's peak reduces and vice-versa happens when sigma is reduced

d. The off-diagonal matrix shows the relation between different variables and skew our gaussian plots

8. Using MV Gaussian for anomaly detection:

a. Steps:



Multivariate Gaussian (Normal) distribution

Parameters μ, Σ $\mu \in \mathbb{R}^n$ $\Sigma \in \mathbb{R}^{n\times n}$

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

Parameter fitting:

Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $x \in \mathbb{R}^n$

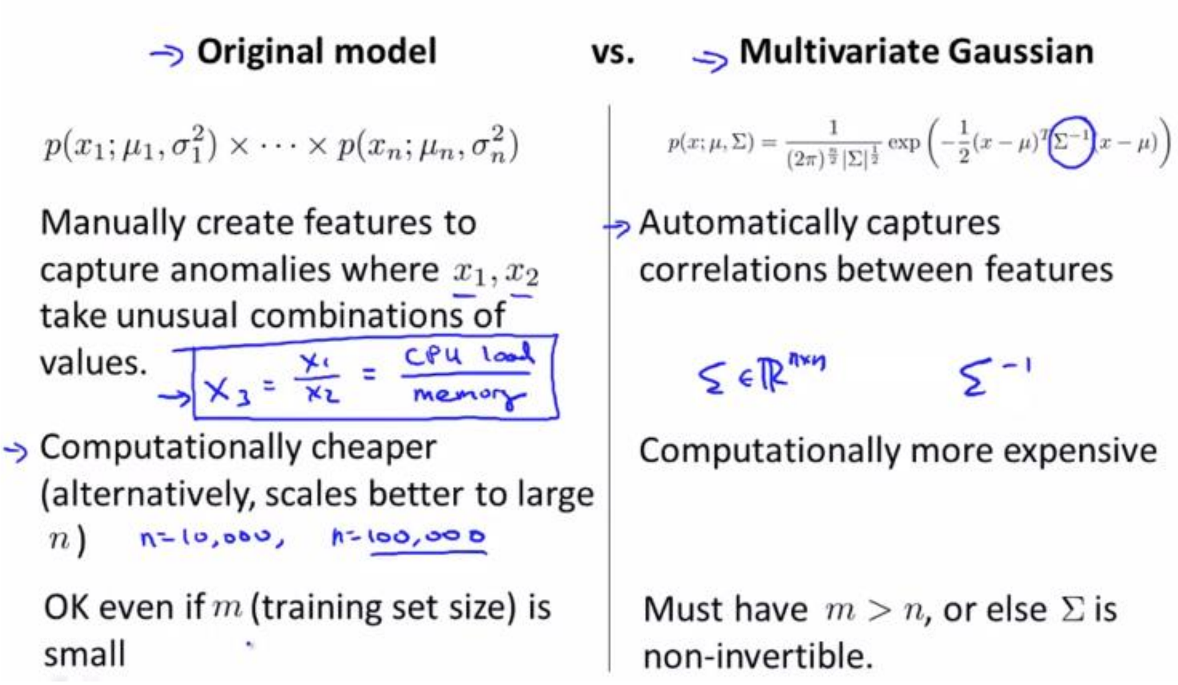
$$\boxed{\mu} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad \boxed{\Sigma} = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)(x^{(i)}-\mu)^T$$

i. Fit model parameters using formulas above

ii. Given new example x_test, find p(x_test) using MV Gaussian formula and compare with threshold value

b. When the off-diagonal elements (correlation between variables) is 0, the MV Model Gaussian model behaves similar to the old model.

c. MV Model v/s Normal:



Original model vs. **Multivariate Gaussian**

$p(x_1;\mu_1,\sigma_1^2) \times \cdots \times p(x_n;\mu_n,\sigma_n^2)$

Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values. $\boxed{x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}}$

Computationally cheaper (alternatively, scales better to large n) $n=10{,}000$, $n=100{,}000$

OK even if m (training set size) is small

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

Automatically captures correlations between features

$\Sigma \in \mathbb{R}^{n\times n}$ Σ^{-1}

Computationally more expensive

Must have $m > n$, or else Σ is non-invertible.

9. Recommender System:

   a. In this problem, the dataset has following parameters:

      i. nu = number of users

      ii. nm = number of movies

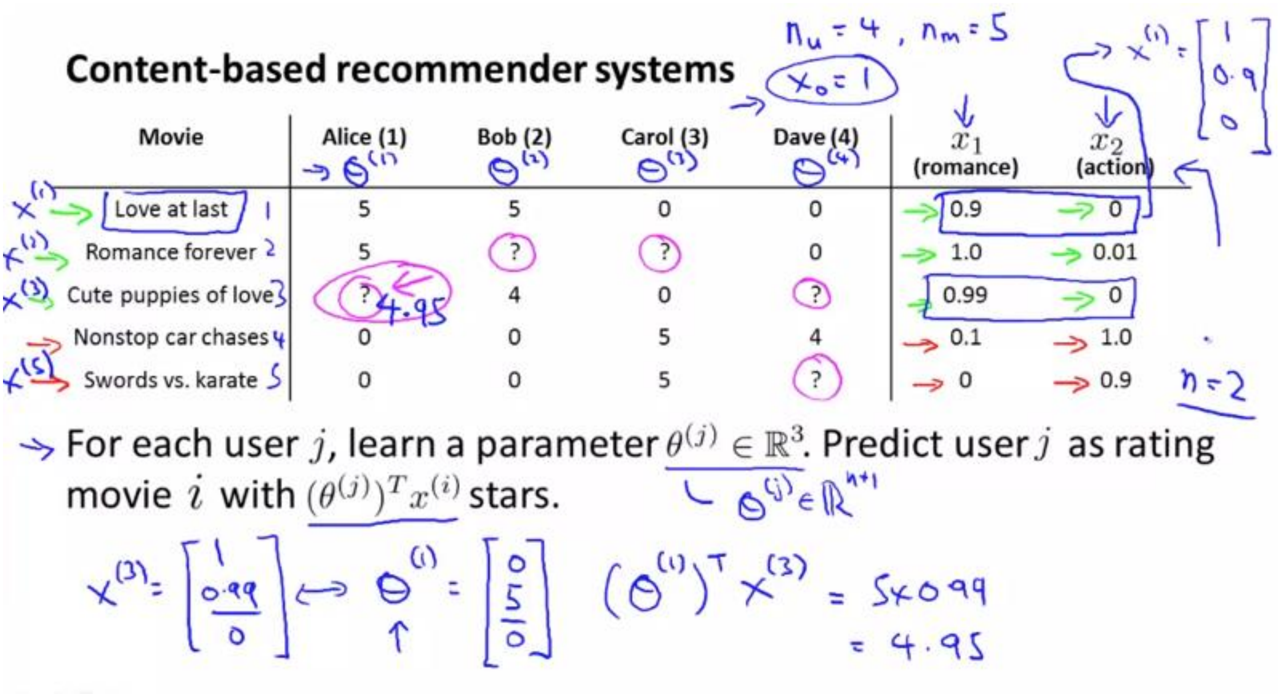
      iii. r(I,j) = 1 if jth user has rated ith movie

      iv. y)I,j) = rating given by jth user to ith movie (exist only when r(I,j) = 1)

   b. Content-based recommendations:

      i. For each movie, we will measure certain set of features (like if it is romance, action, etc)

      ii. For each user, we will learn a hypothesis with parameters theta and then use the previous ratings along with theta to predict future ratings y



Content-based recommender systems

$n_u = 4$, $n_m = 5$

$x_0 = 1$

$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$ Love at last 1	5	5	0	0	0.9	0
$x^{(2)}$ Romance forever 2	5	?	?	0	1.0	0.01
$x^{(3)}$ Cute puppies of love 3	? 4.95	4	0	?	0.99	0
Nonstop car chases 4	0	0	5	4	0.1	1.0
$x^{(5)}$ Swords vs. karate 5	0	0	5	?	0	0.9

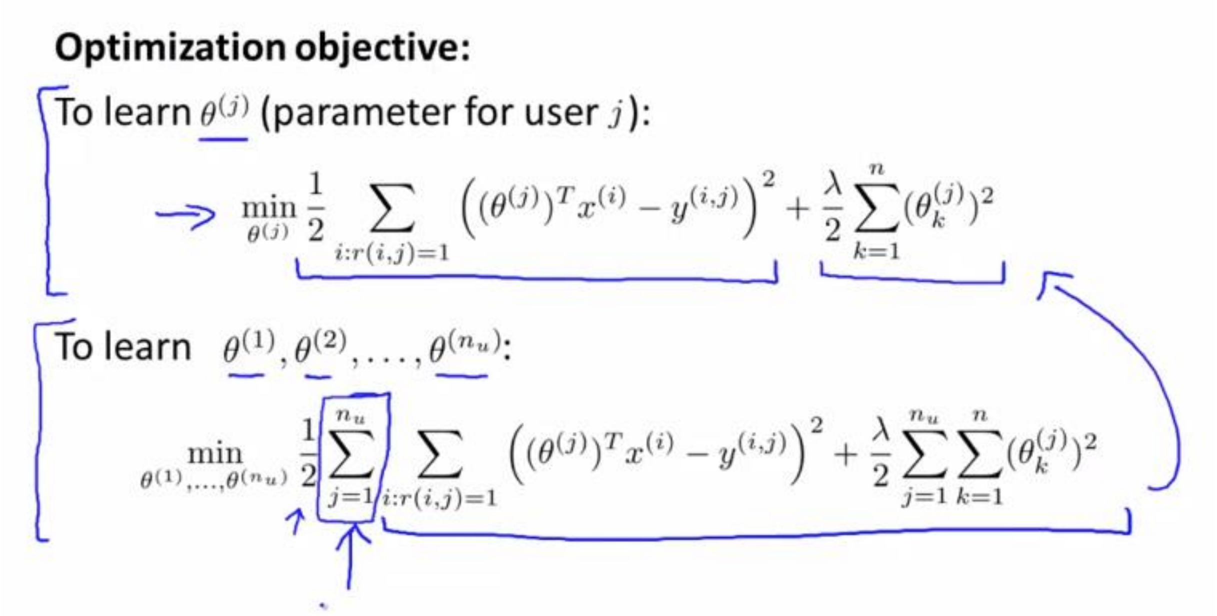
$n = 2$

→ For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

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

$$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} \longleftrightarrow \theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \qquad (\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$$

iii. Theta j for each user can be learned as shown (similar to linear regression):



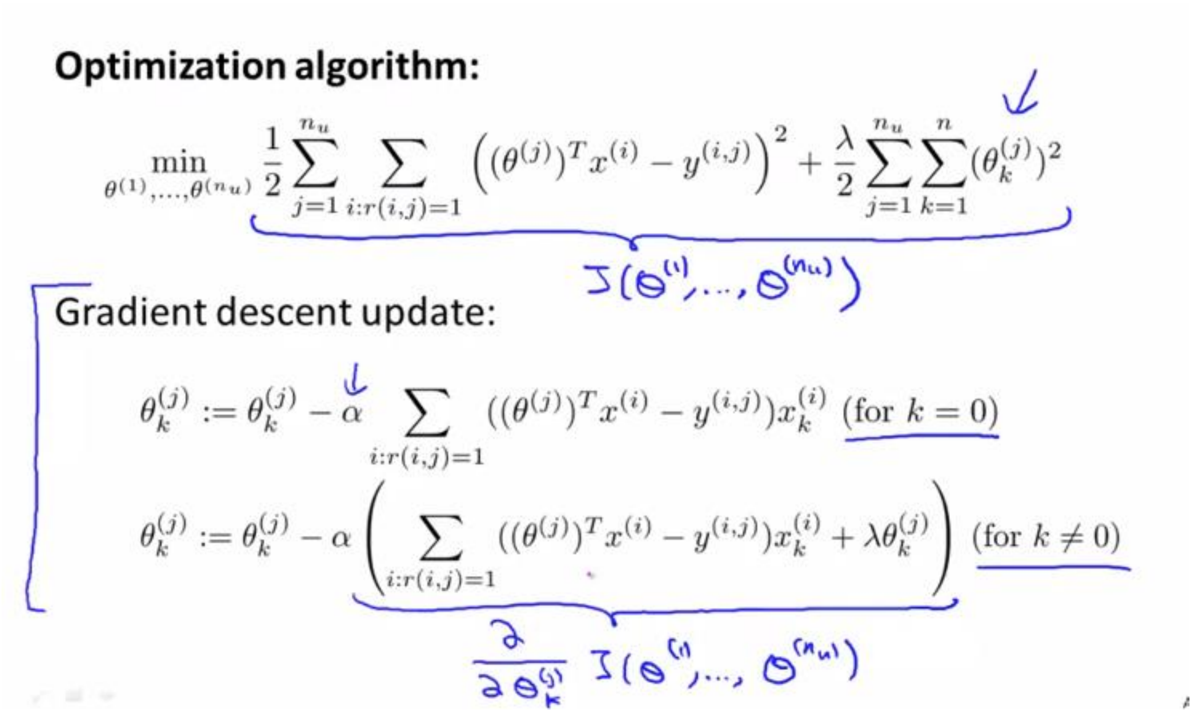
Optimization objective:

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

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

To learn $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)}$:

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



Optimization algorithm:

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

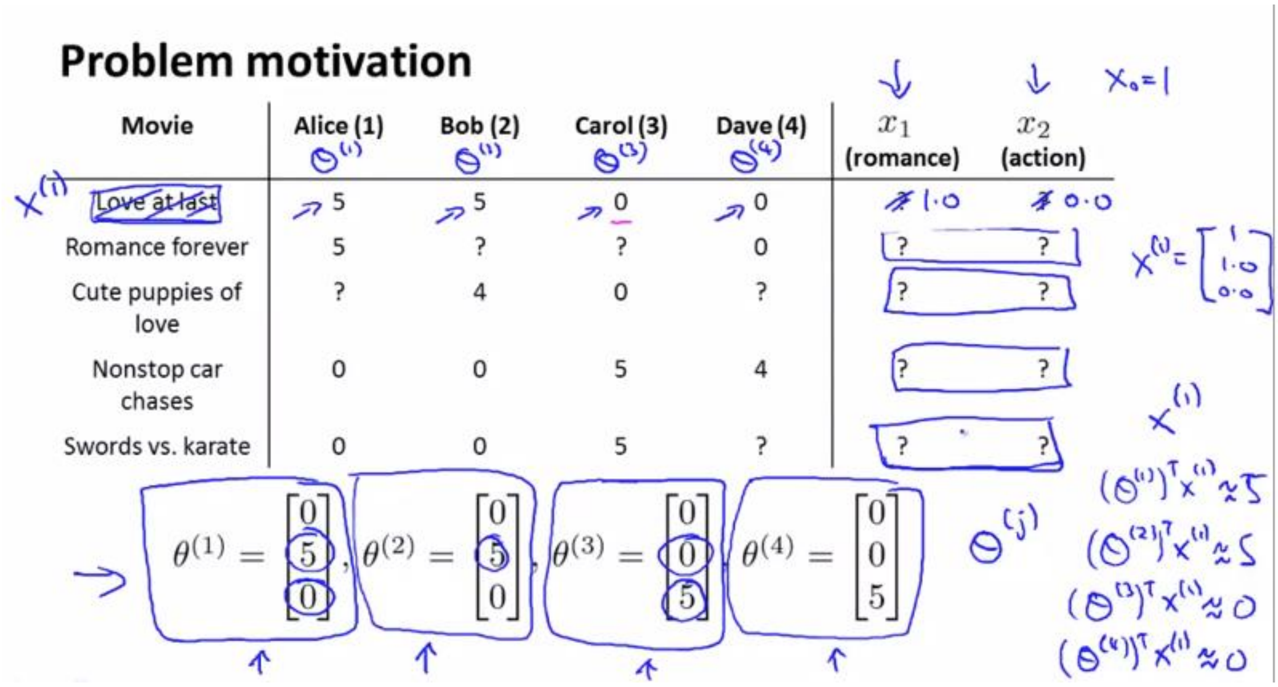
Gradient descent update:

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

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \underbrace{\left(\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda\theta_k^{(j)}\right)}_{\frac{\partial}{\partial\theta_k^{(j)}} J(\theta^{(1)},\dots,\theta^{(n_u)})} \quad \underline{(\text{for } k \neq 0)}$$

iv. Using grad and J like earlier, we can also use some advanced optimization algorithm other than gradient descent.

10. Collaborative Filtering:
    a. Algorithm will implement feature learning and use best features
    b. How to infer features of movies (x) from taking survey of users (to find theta for each user)?



Problem motivation

$x_0 = 1$

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$ ~~Love at last~~	5	5	0	0	~~?~~ 1.0	~~?~~ 0.0
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

$$x^{(1)} = \begin{bmatrix} 1 \\ 1.0 \\ 0.0 \end{bmatrix}$$

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

$\theta^{(j)}$ $x^{(1)}$

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

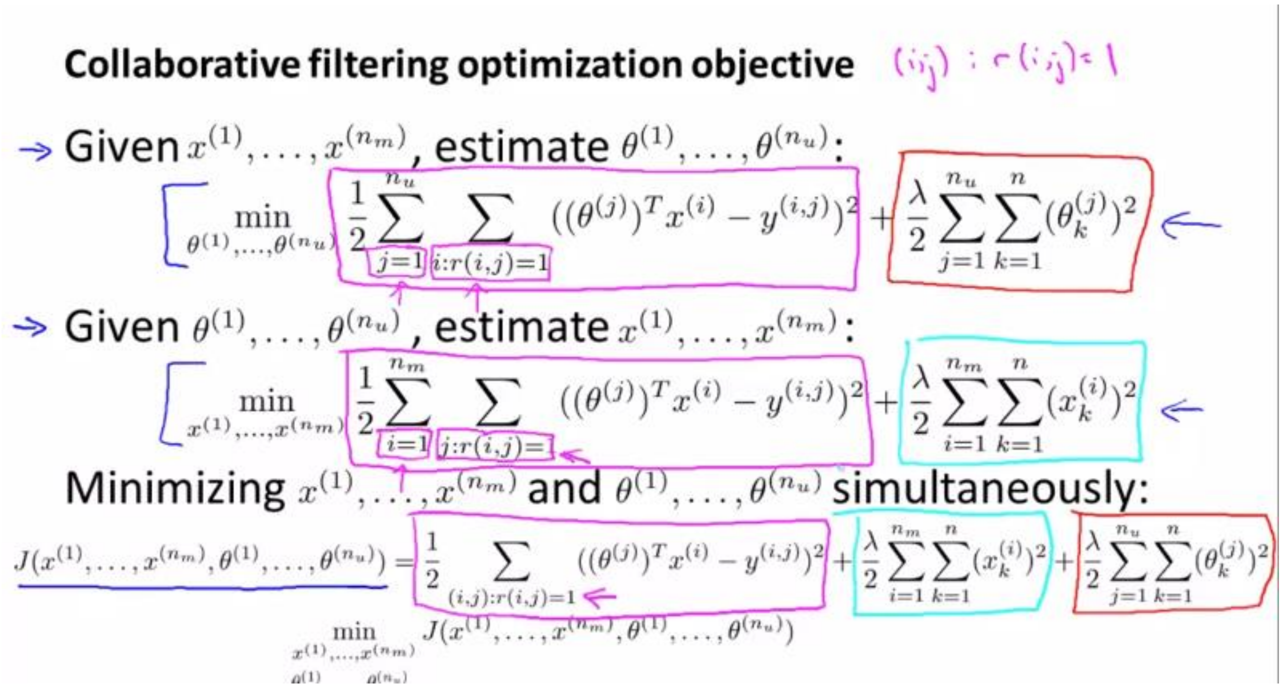
$(\theta^{(2)})^T x^{(1)} \approx 5$

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

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

c. There are 2 scenarios possible:
   i. Given theta from users → we find x for movies
   ii. Given x for movies → we find theta for users
d. Approach we can use is guess theta → find X → refine theta → find X → ….. and so on
e. This method works because we have multiple movies being rated by multiple users

11. Collaborative Filtering: Algorithm
    a. Objectives can be the following two:
    b. Eventually we will merge both the objectives and state it as follows:



Collaborative filtering optimization objective

$(i,j):r(i,j)=1$

→ Given $x^{(1)},\dots,x^{(n_m)}$, estimate $\theta^{(1)},\dots,\theta^{(n_u)}$:

$$\left[\min_{\theta^{(1)},\dots,\theta^{(n_u)}} \boxed{\frac{1}{2}\sum_{\boxed{j=1}}^{n_u}\sum_{\boxed{i:r(i,j)=1}}((\theta^{(j)})^Tx^{(i)}-y^{(i,j)})^2}+\boxed{\frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^{(j)})^2} \leftarrow\right.$$

→ Given $\theta^{(1)},\dots,\theta^{(n_u)}$, estimate $x^{(1)},\dots,x^{(n_m)}$:

$$\left[\min_{x^{(1)},\dots,x^{(n_m)}} \boxed{\frac{1}{2}\sum_{\boxed{i=1}}^{n_m}\sum_{\boxed{j:r(i,j)=1}}((\theta^{(j)})^Tx^{(i)}-y^{(i,j)})^2}+\boxed{\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2} \leftarrow\right.$$

Minimizing $x^{(1)},\dots,x^{(n_m)}$ and $\theta^{(1)},\dots,\theta^{(n_u)}$ simultaneously:

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

$$\min_{\substack{x^{(1)},\dots,x^{(n_m)}\\ \theta^{(1)},\dots,\theta^{(n_u)}}} J(x^{(1)},\dots,x^{(n_m)},\theta^{(1)},\dots,\theta^{(n_u)})$$

c. In this new version, we do not need to go back and forth between theta and x optimization, we minimize both at the same time

d. Steps of Algo:

   i. Initialize x and theta to random values

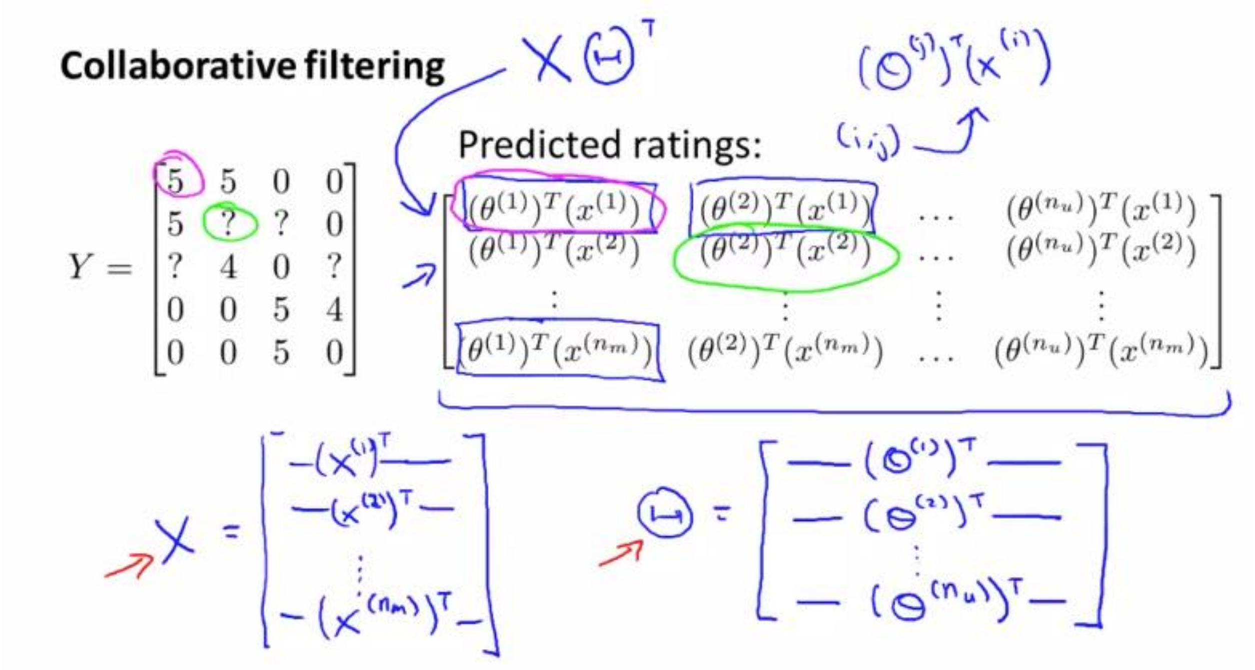
   ii. Minimize J (x, theta) using gradient descent/ advanced optimization algorithm

   iii. For every movie and user, update respective x and theta

   iv. For jth user with parameters theta and a movie with features x, predict theta'*x

12. Vectorization: Low Rank Matrix Factorization:

a. Matrix representation of y and predictions is as follows:



Collaborative filtering

$X\,\Theta^T$

$(\theta^{(j)})^T(x^{(i)})$

(i,j)

Predicted ratings:

$$Y=\begin{bmatrix}5 & 5 & 0 & 0\\ 5 & ? & ? & 0\\ ? & 4 & 0 & ?\\ 0 & 0 & 5 & 4\\ 0 & 0 & 5 & 0\end{bmatrix}$$

$$\underbrace{\begin{bmatrix}(\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)})\\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)})\\ \vdots & \vdots & \vdots & \vdots\\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)})\end{bmatrix}}$$

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

b. This is also called low rank matrix factorization

c. Usually features learned by algorithm will be meaningful

d. How to find movie j related movie i? (similarity between movies)

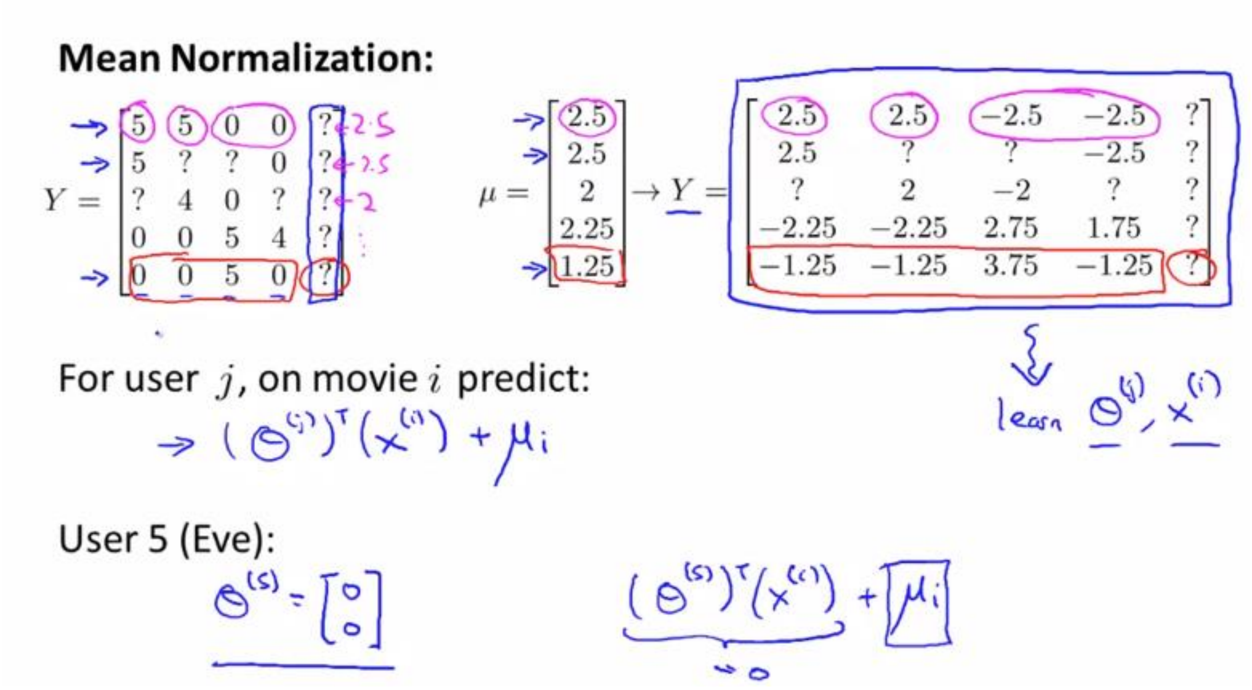
   i. When || xi – xj || is small, they are similar

13. Recommendation: Mean Normalization:

a. When we have a user with no movie records, the theta as per algorithm will be 0. This won't help us in predicting future movies for them.

b. Therefore, we implement mean normalization to overcome this challenge.

c. It can be done by finding mean rating of every movie and subtract that from all user’s rating



Mean Normalization:

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \begin{matrix} \leftarrow 2.5 \\ \leftarrow 2.5 \\ \leftarrow 2 \\ \\ \\ \end{matrix} \qquad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow \underline{Y} = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

learn $\underline{\Theta^{(j)}}, \underline{x^{(i)}}$

For user j, on movie i predict:

$$\rightarrow (\Theta^{(j)})^T(x^{(i)}) + \mu_i$$

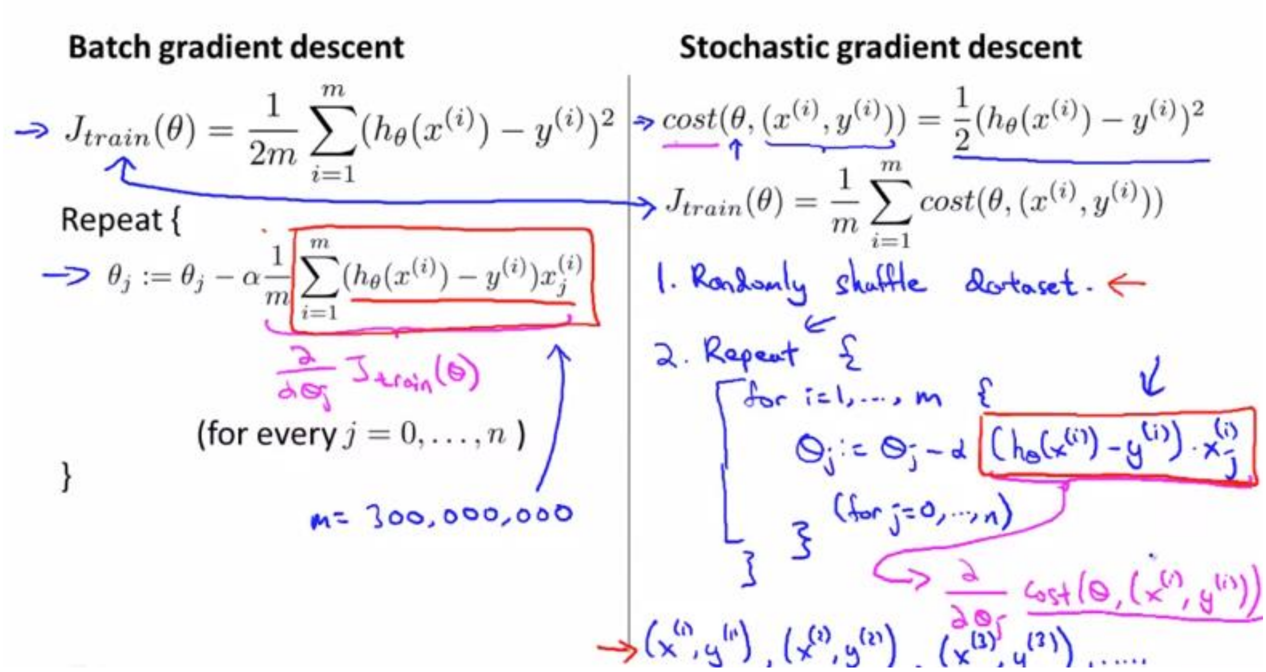
User 5 (Eve):

$$\underline{\Theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}} \qquad \underbrace{(\Theta^{(5)})^T(x^{(i)})}_{=0} + \boxed{\mu_i}$$

d. Similarly, we can apply normalization in situation when a particular does not have any rating

Week 10:

1. Large Scale ML: Large Datasets
    a. One way to get a good ML model is to use a low bias model
    b. Large data has computation problems
2. Stochastic Gradient Descent:
    a. Normal Gradient Descent (GD) will be computationally very expensive for large datasets
    b. Batch GD algorithm (the one used normally used) will give minimized thetas after iterating over all examples. This step is to be reduced as it leads to computational complexity for large dataset.



Batch gradient descent

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

```
Repeat {
    $\theta_j := \theta_j - \alpha \frac{1}{m} \boxed{\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}}$
    $\frac{\partial}{\partial\theta_j} J_{train}(\theta)$
        (for every $j = 0,\ldots,n$)
}
```

$m = 300{,}000{,}000$

Stochastic gradient descent

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

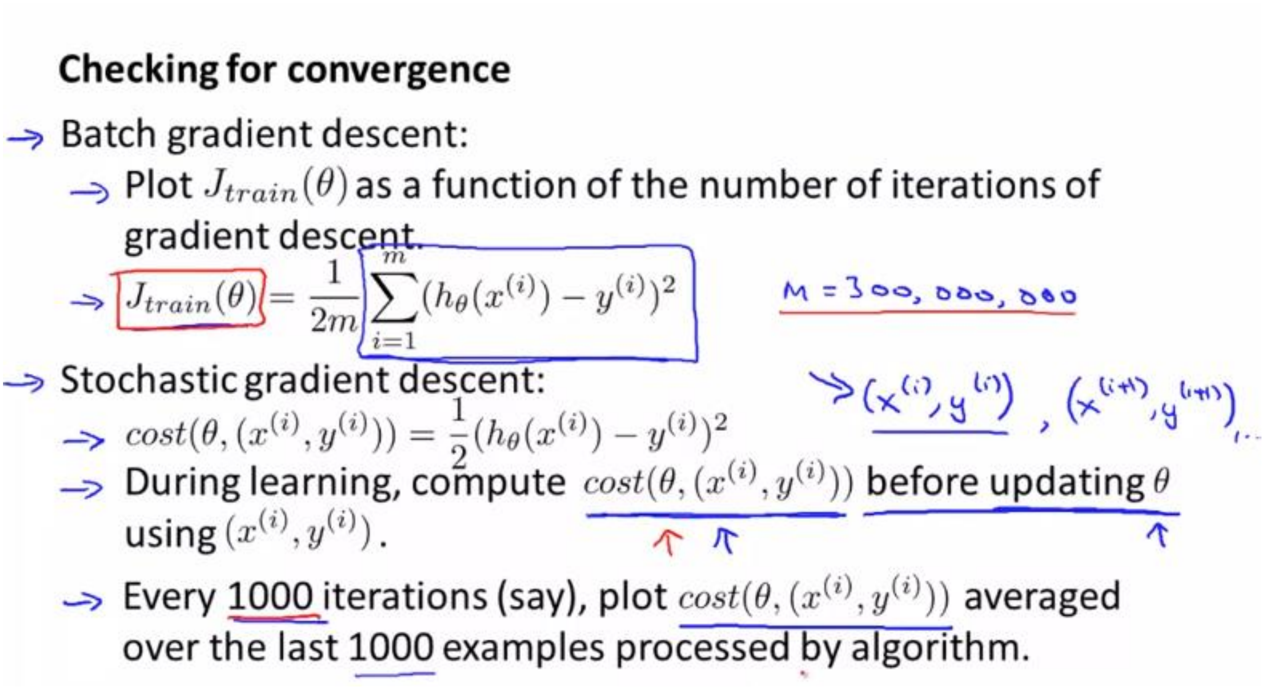
$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta,(x^{(i)},y^{(i)}))$$

```
1. Randomly shuffle dataset.
2. Repeat {
    for i=1,...,m {
        $\theta_j := \theta_j - \alpha \boxed{(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}$
            (for j=0,...,n)
    }
}
```

$\frac{\partial}{\partial\theta_j} cost(\theta,(x^{(i)},y^{(i)}))$

$(x^{(1)},y^{(1)}), (x^{(2)},y^{(2)}), (x^{(3)},y^{(3)}) \ldots\ldots$

    c. In stochastic GD, each iteration just fits one example better. That allows rapid convergence.
    d. Stochastic GD will generally move the parameters towards global minimum but not always (because there is no mechanism for finding a minimum among cost function values for all examples)
    e. It also does not converge like GD, it keeps moving around global minimum
    f. Repeat of outer loop (usually 1 to 10) will depend on dataset.
3. Mini-batch GD
    a. It can work faster than batch GD
    b. Here we change theta based on mini-batches unlike update on every iteration like in stochastic
    c. We look at b examples instead 1 example like Stochastic GD because it can be properly implemented in vectorized way and allow faster computation (through parallel computing)
    d. The disadvantage is choice of b (usually 10 or from 2 to 100).
4. Stochastic Gradient Descent Convergence
    a. GD convergence can be visualized as follows for both cases:



Checking for convergence

- Batch gradient descent:
  - Plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent.
  - $\boxed{J_{train}(\theta)} = \frac{1}{2m}\boxed{\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2}$

$M = 300{,}000{,}000$

- Stochastic gradient descent:
  - $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$
  - During learning, compute $cost(\theta, (x^{(i)}, y^{(i)}))$ before updating θ using $(x^{(i)}, y^{(i)})$.
  - Every 1000 iterations (say), plot $cost(\theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples processed by algorithm.

$(x^{(i)}, y^{(i)}), (x^{(i+1)}, y^{(i+1)}), \ldots$

    b. To make theta converge for stochastic gradient, we must decrease learning rate alpha over each iteration.

5. Online Learning:
    a. When data stream is continuously coming and we need to learn from that
    b. We take example, learn from it (update theta of hypothesis) and then discard and repeat this process infinitely.
6. Map Reduce and Data Parallelism:
    a. Computation is divided among multiple processors/ cores
    b. When sum over large iteration is a part of any algorithm, it can be broken down by map reduce

Week 11:

1. Photo OCR: Problem description and pipeline:
    a. Read texts in an image so that images become searchable
    b. OCR from scanned documents is easy but from photographs is still difficult
    c. It can help in car navigation by reading road signs
    d. The pipeline of photo OCR is as follows:
        i. Text detection
        ii. Character segmentation
        iii. Classify characters
        iv. Spelling correction (optional step)
    e. Image → Text Detection → Character Segmentation → Character Recognition
2. Sliding Windows:
    a. For text detection, we will start with pedestrian detection system assuming aspect ratio constant
        i. Gather positive and negative pedestrian examples of the aspect ratio value fixed earlier
        ii. Then train a NN or similar classification algorithm to classify an image patch if it is a pedestrian or not
        iii. Iterate over the entire test image and then classify these patches. Use proper step size/ slide (sliding window value) for better performance
        iv. After this, iterate again with larger image patch size (resize it to our fixed dimensions)
    b. Text Detection:
        i. Gather data of text positive and negative
        ii. Train a classifier
        iii. We will then apply expansion to the positively classifier patches. That means that if gap between texts patches is within threshold, we will make the gap also as part of text
        iv. We then eliminate the patches whose aspect ratio is funny
    c. Character Segmentation:
        i. Then we will gather data that has positive label as split/ gap between characters
        ii. Train a classifier on this dataset, that can classify midpoint between characters/ split
        iii. This will be 1D sliding window (horizontally)
        iv. Then we apply it over each image patches and then make split images in individual characters
    d. Character Classification/ Recognition:
        i. Apply multiclass classification
3. Getting lots of data and artificial data:
    a. Ideal method is to use a low biased algorithm with a very large dataset
    b. Artificial Data Synthesis (It is an art):
        i. Create new data from scratch
        ii. Small labelled data to large by distortion/ augmentation to amplify dataset
    c. Distortion added in data must be representative of the noise/ distortion that could actually be present in test set. Adding random/ Gaussian White noise/ meaningless noise is less likely to be useful.
    d. Steps:
        i. Verify low bias classifier by plotting learning curves. You can keep adding features/ number of hidden units of NN until you get a low bias classifier
        ii. Is it easy to get 10 x of current dataset?
            1. Collect yourself
            2. Artificial data synthesis
            3. Crowd source
4. Ceiling Analysis (we get an upper bound on what performance improvement we get if one phase of pipeline gives ideal performance):
    a. We should have a single number evaluation parameter
    b. Best way to allocate time is find which part of pipeline on giving perfect test set (100% accurate) leads to most improvement in the final output