Machine Learning

Week 1

▼ Introduction

▼ Supervised Learning vs Unsupervised Learning

Supervised Learning

Classification, Regression

Unsupervised Learning

Clustering, Cocktail Problem(Isolating Sound)

▼ Model and Cost Function

▼ Model Representation

Supervised Learning gives the "right answer" by predicting based on past experience(Training From Training Set)

▼ Notation List

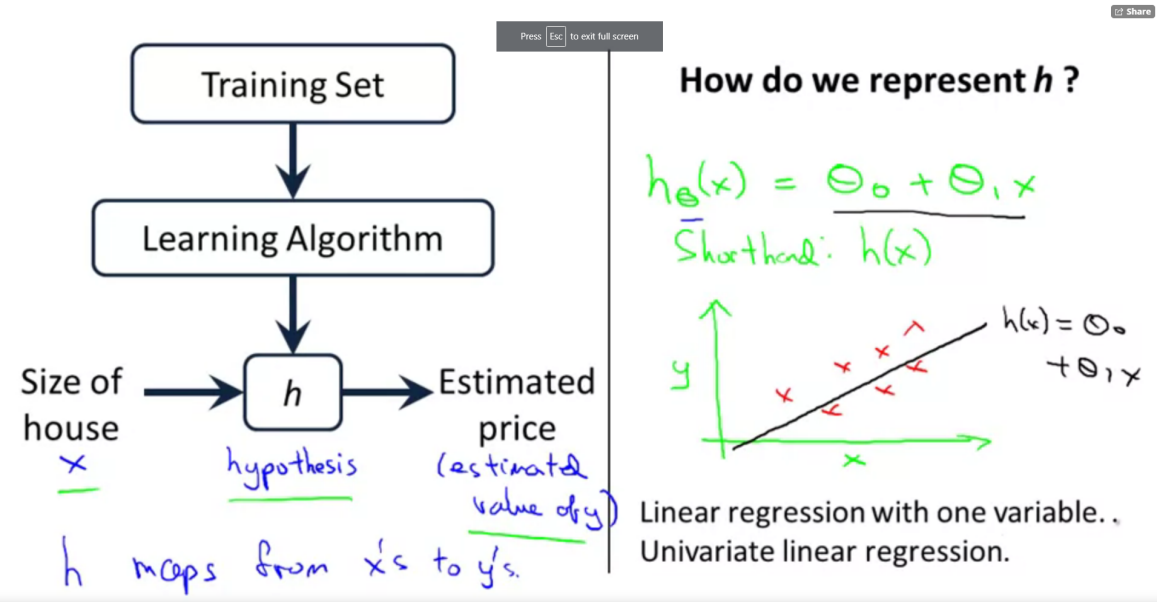
m = Number of Training Examples

x = input features

y = output/ targeted variable

(x, y) = one training example

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

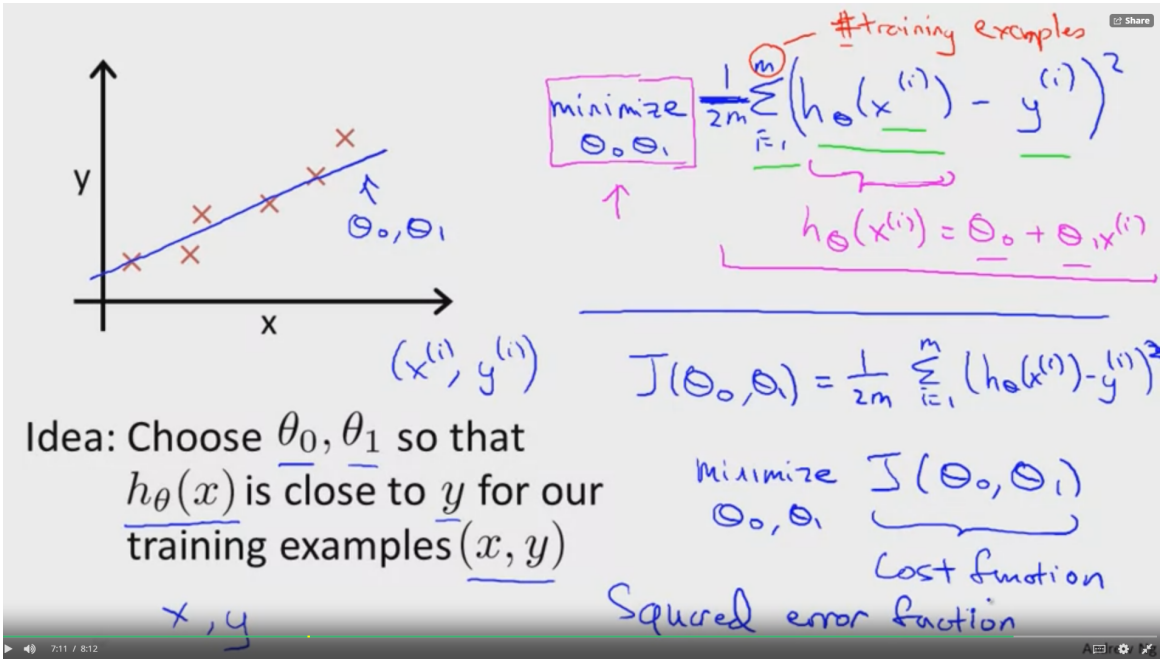


^ h(hypothesis<function>) maps x to y in supervised learning

$h_\theta(x) = \theta_0 + \theta_1 x$: y is Linear function of x : Linear Regression(Univariate<one var> Linear Regression)

▼ Cost Function

Squared Error Function



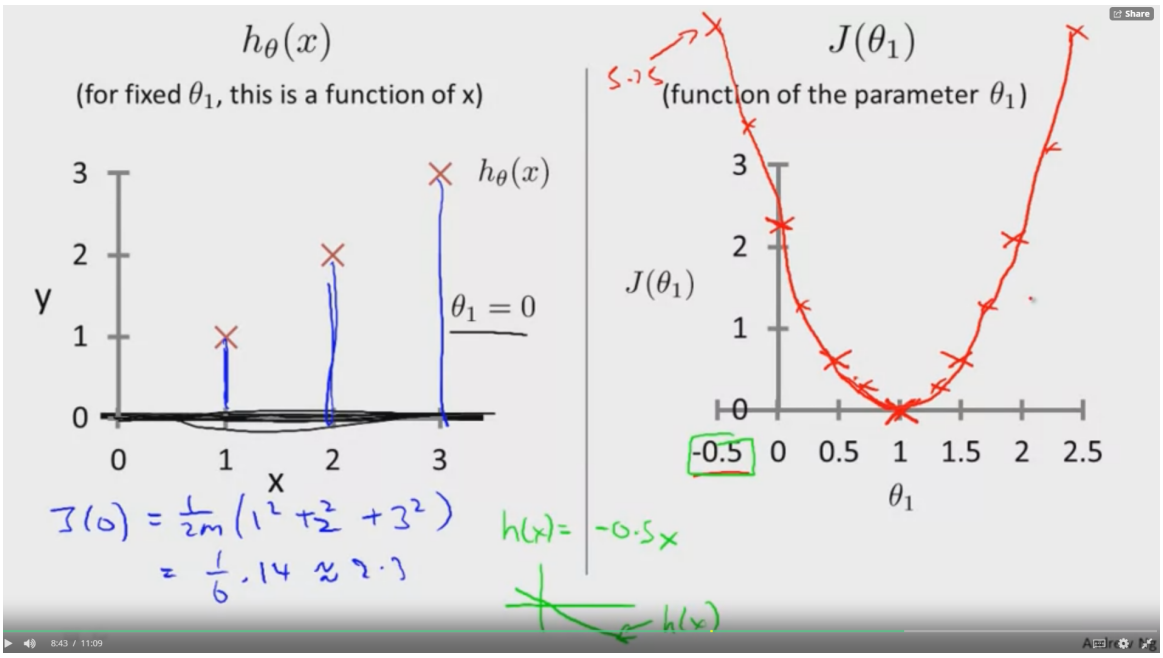
θ_0, θ_1 = parameters

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

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

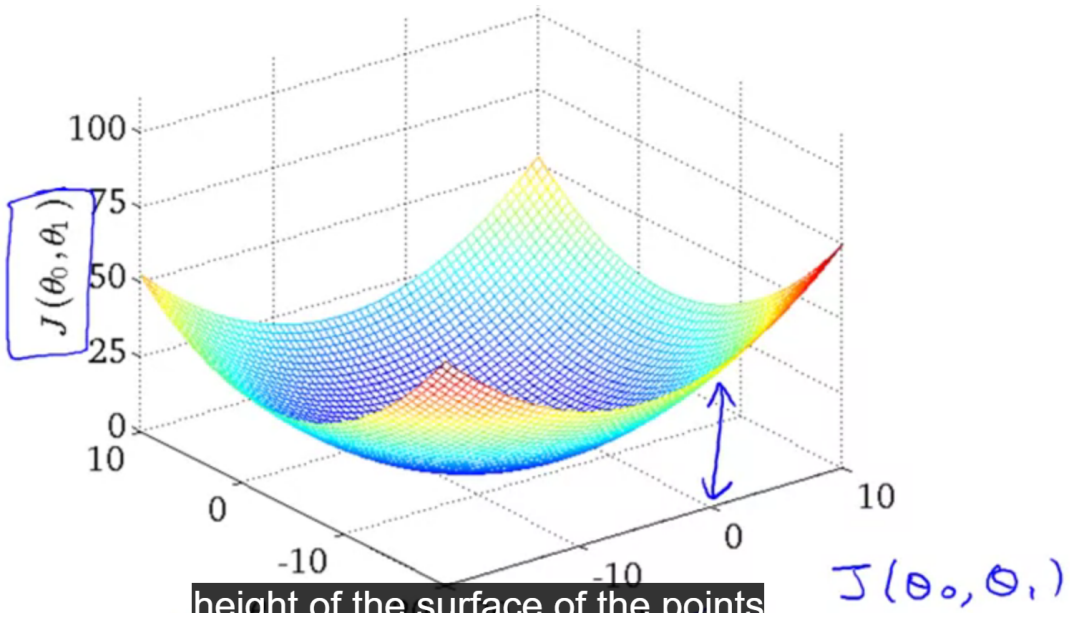
minimize Cost : $J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y)^2$

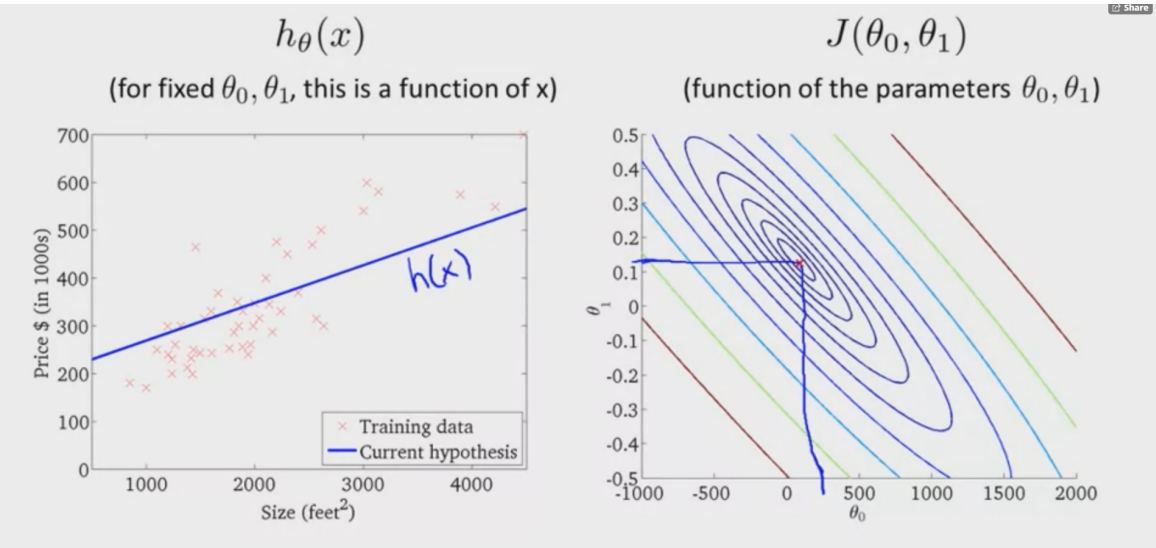
- Commonly used for regression problem



Changes parameter θ_1 such that $J(\theta_1)$ is minimized

Contour Plot/Figures (3D Plot)





θ_0, θ_1 corresponds to x-axis and y-axis

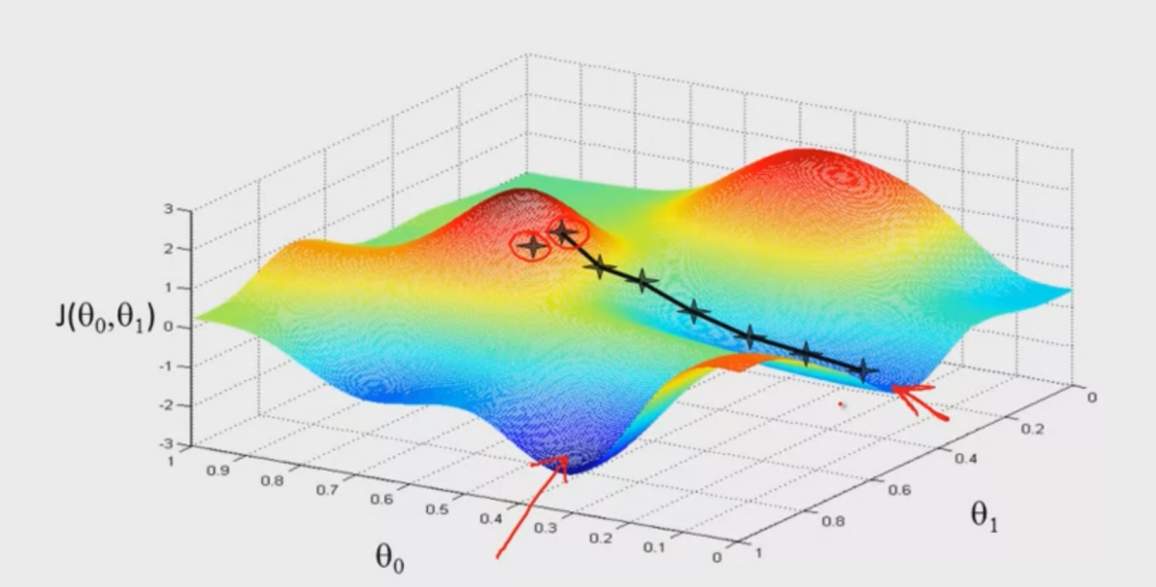
$J(\theta_0, \theta_1)$ corresponds to height/ Oval line

▼ Parameter Learning

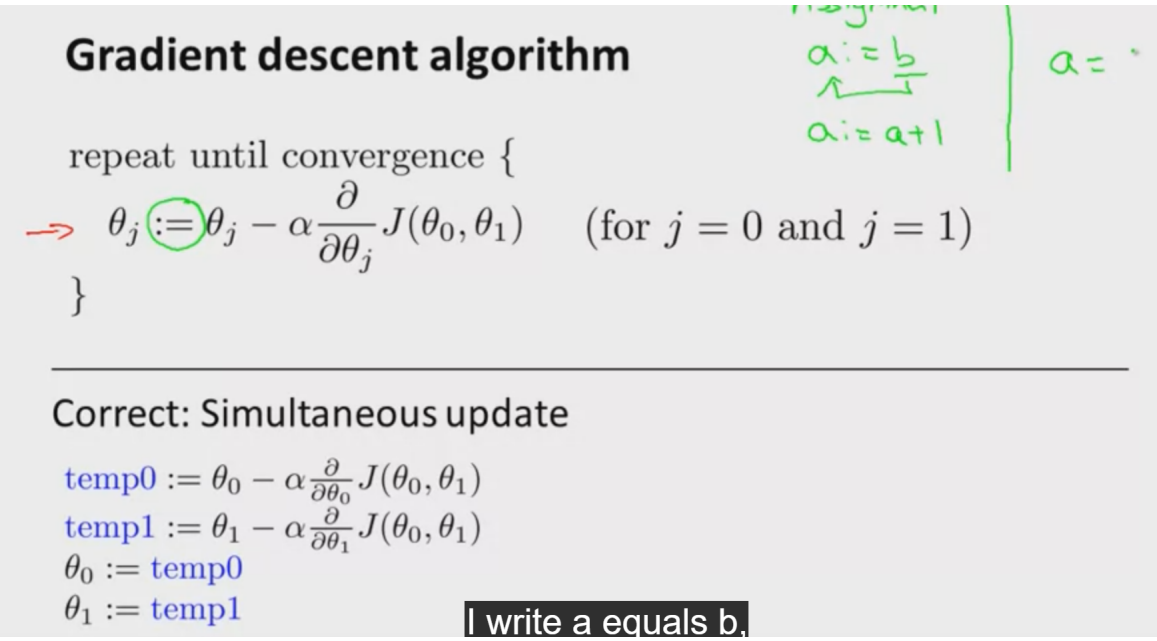
▼ Gradient Descent

Start with some θ_0, θ_1

Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ and end up at a local minimum



> Starting at different point may end up (for diff cost func) with different Local Minimum!



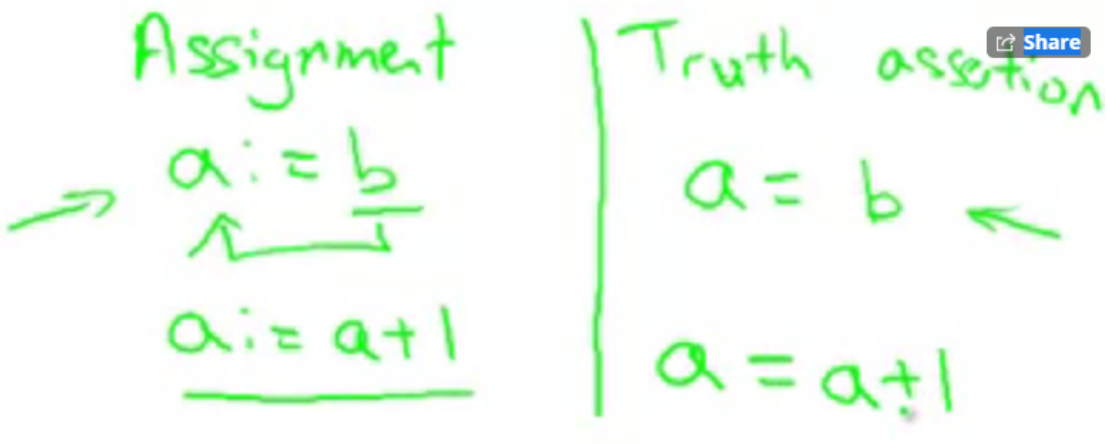
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$\alpha = \text{learning rate}$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = d\theta_j = \text{Derivative of Loss respect to parameter} : \theta_j$$

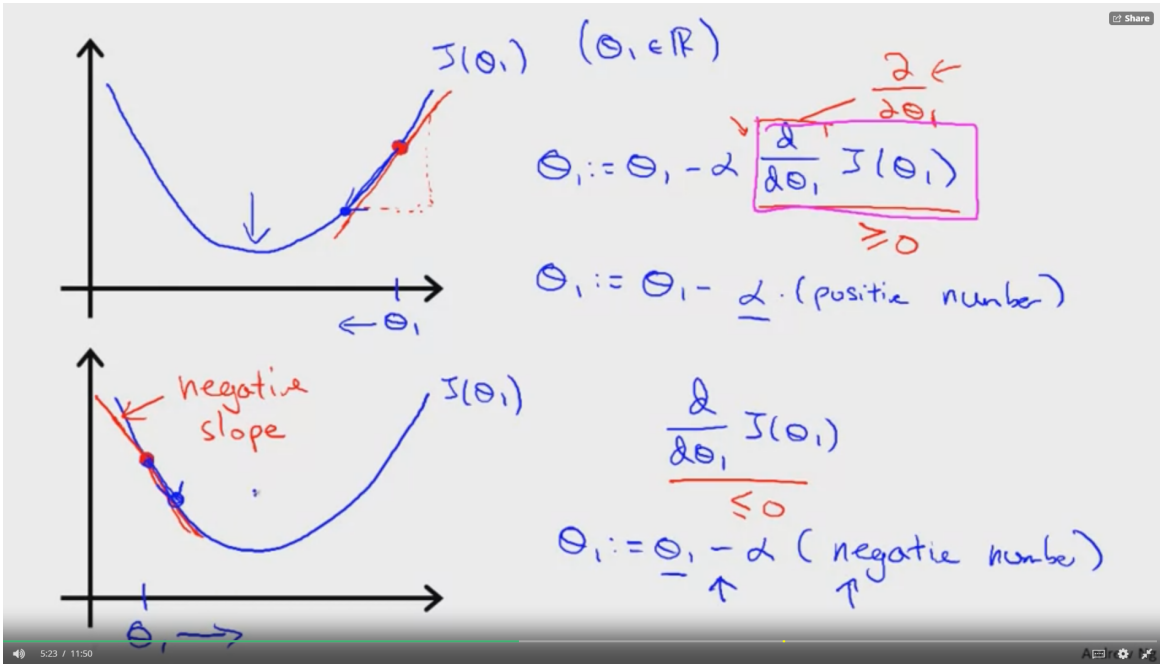
Careful of Simultaneous Update (e.g. Update θ_0 and θ_1 together) of Parameters but not in sequence (e.g. Update θ_0 then only θ_1)

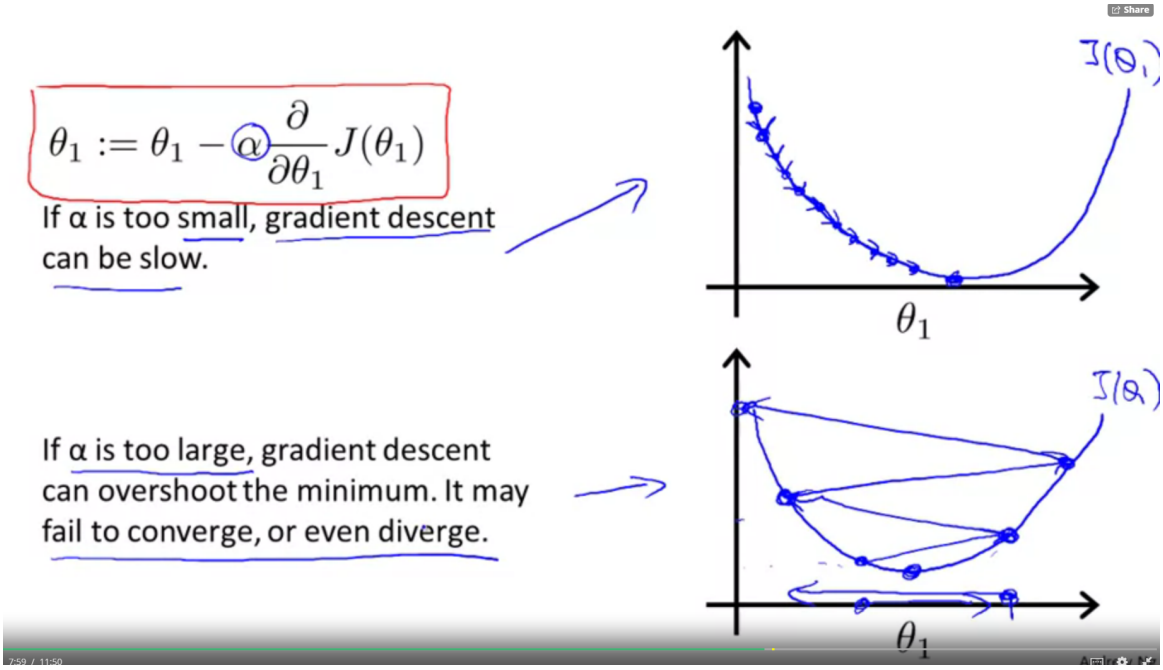
▼ Assignment vs Truth Assertion

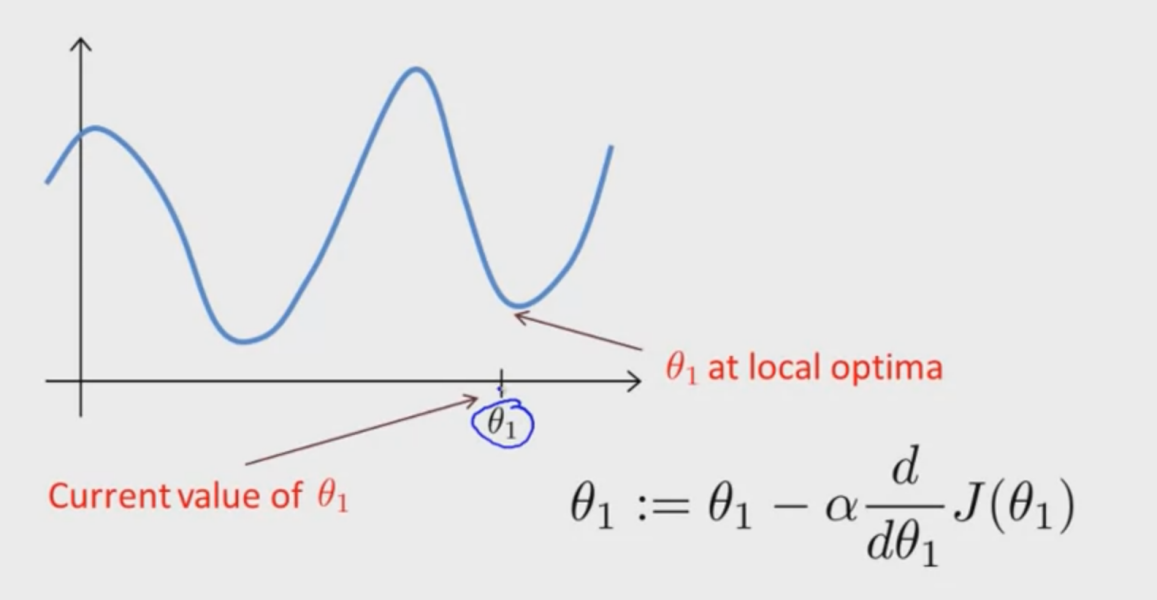


- := is used to assign value like "=" in programming
- = is used to assert truth like "==" in programming

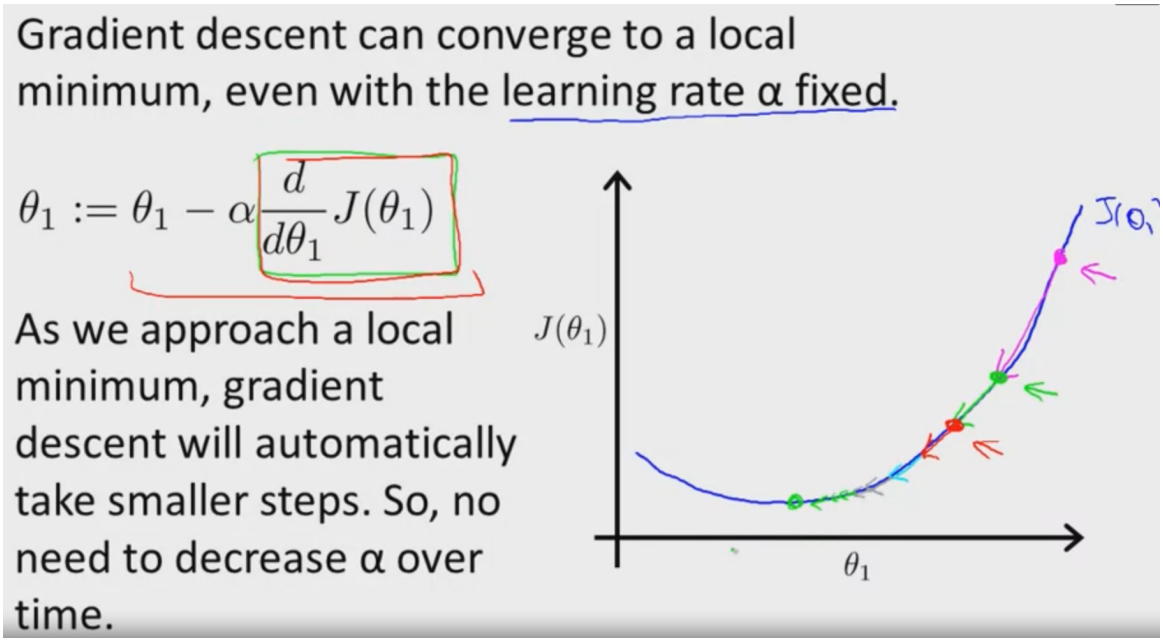
▼ Gradient Descent Intuition







💡 If At local Optima, d1 is 0, parameter would not change



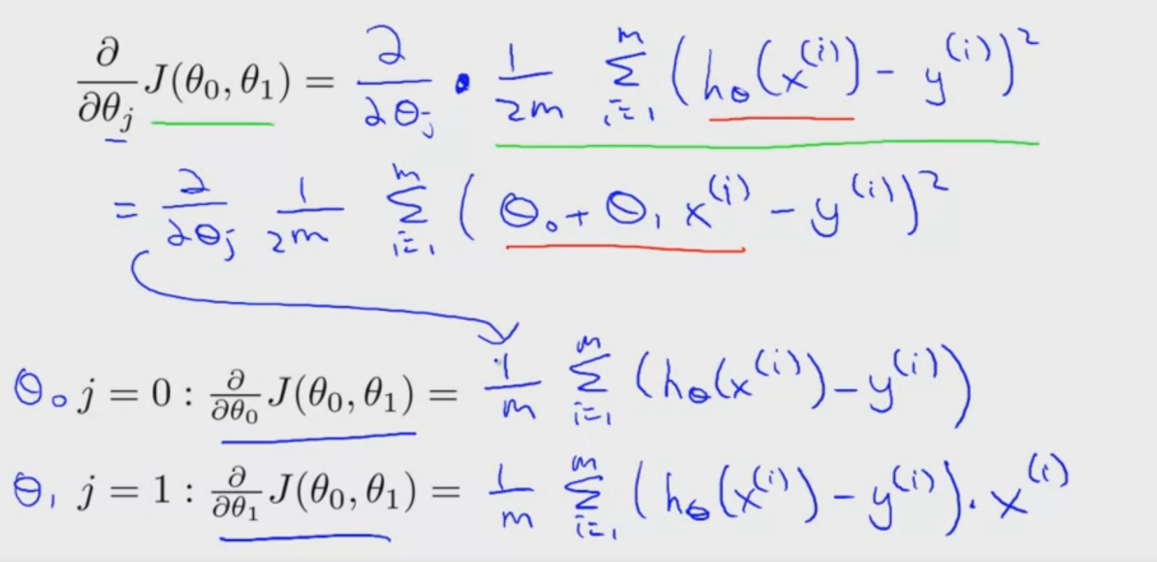
💡 Change in magnitude of derivative will minimize the steps taken in order to converge to local minimum

▼ "Batch" Gradient Descent for Linear Regression

💡 "Batch" Gradient Descent Referring to each step uses all training examples

Gradient descent algorithm	Linear Regression Model

repeat until convergence {
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
(for $j = 1$ and $j = 0$)
}
	$h_\theta(x) = \theta_0 + \theta_1 x$
	$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$



$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} * \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^{(i)})^2 \text{ For j} = 0, 1$$

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

$\theta_1(\text{w/m}) : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^{(i)}) * x^{(i)}$

Gradient descent algorithm

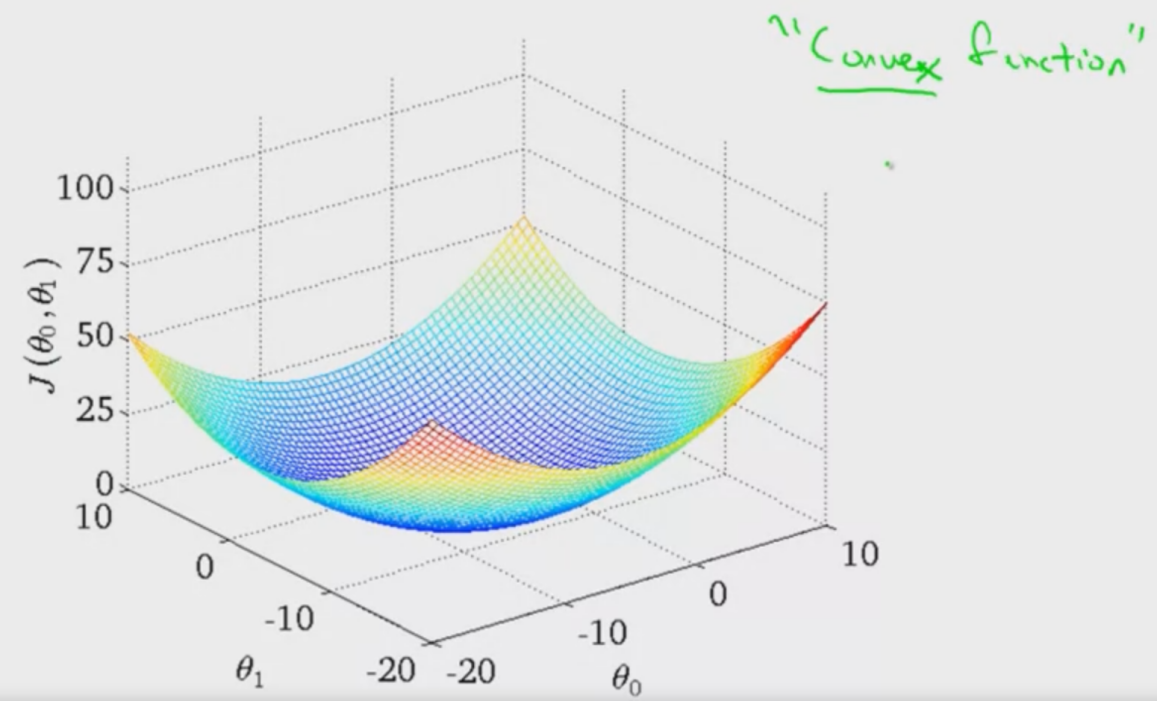
repeat until convergence {

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

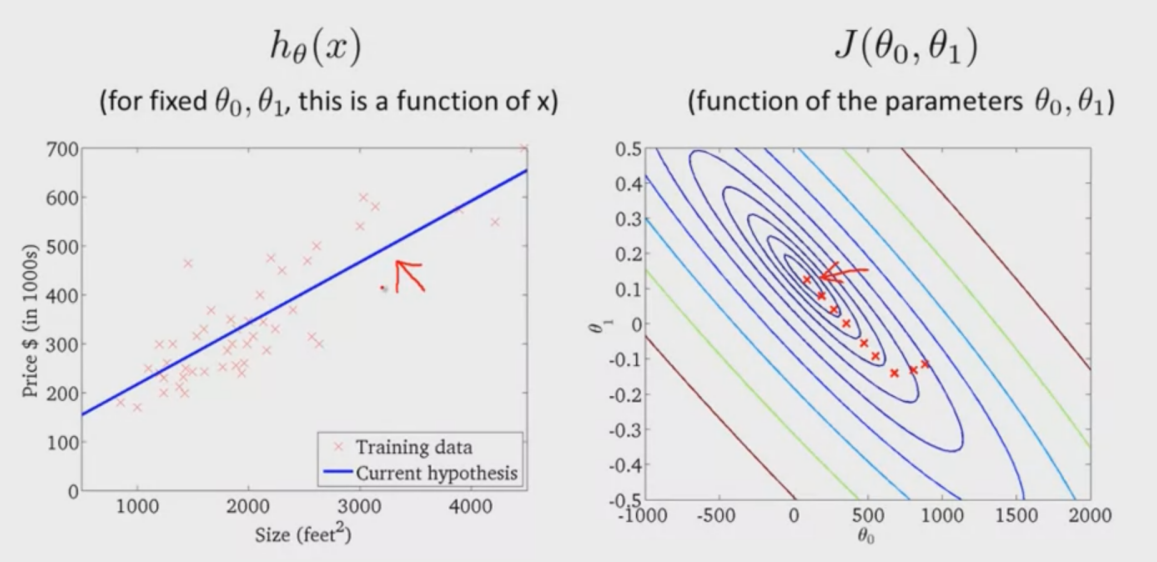
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

update θ_0 and θ_1 simultaneously



💡 Gradient Descent for Squared Error will always end up in global minimum

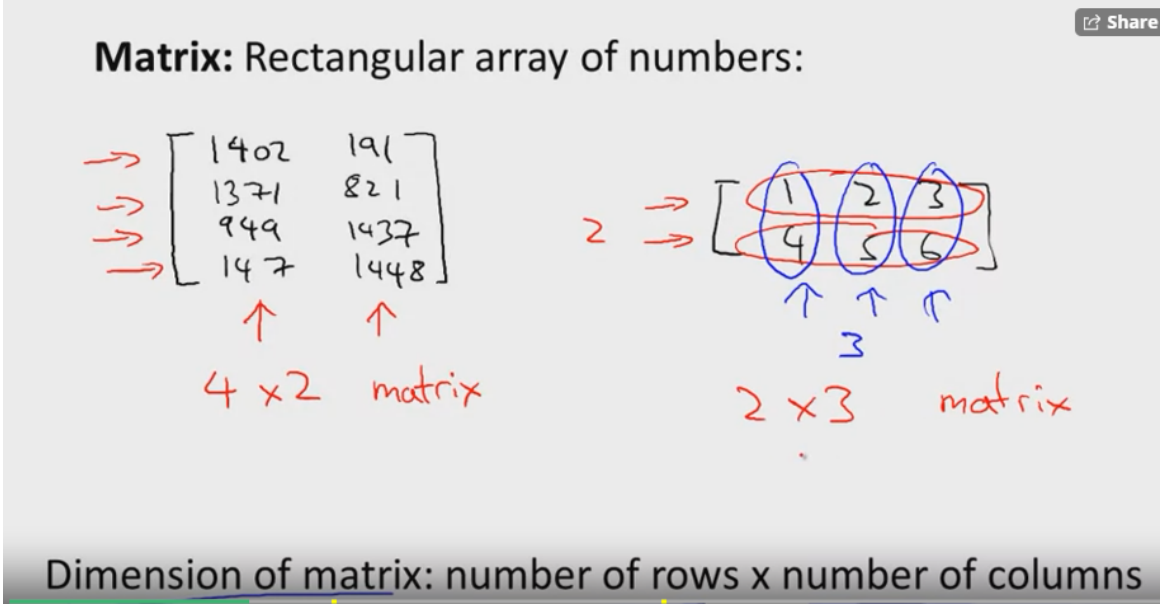


▼ Linear Algebra Review

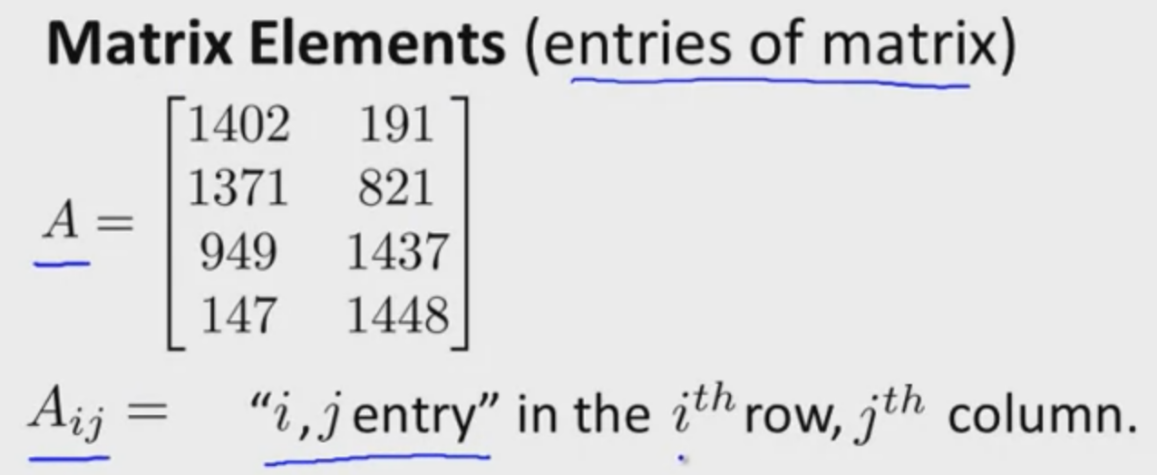
▼ Matrices and Vectors

Matrices

- Upper-Case Notation

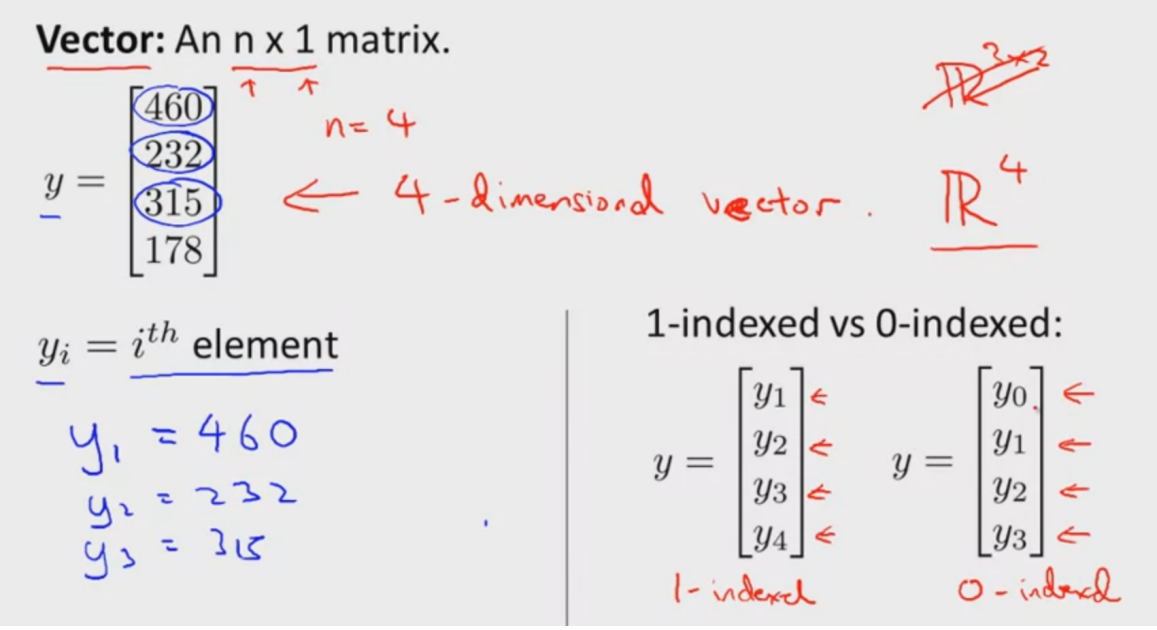


- Matrix is rectangle array of numbers written in squared bracket
- Dimensions = number of rows x number of columns



Vector

- Lower-case notation



- Matrix with one column

^ 4-dimensional vector $\mathbb{R}^4$

$y_i = i^{th}\text{element}$

1-indexed vectors is more common in math, 0-indexed common in prog lang

▼ Addition, Scalar Multiplication, Matrix Vector Multiplication

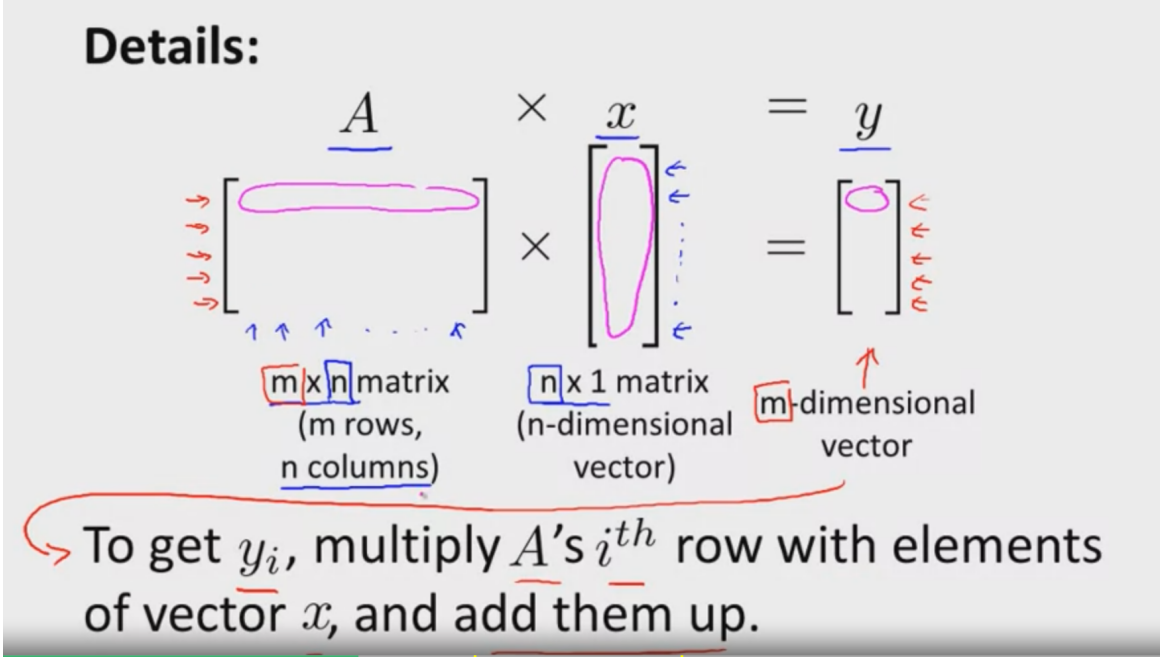
Addition

- Same size

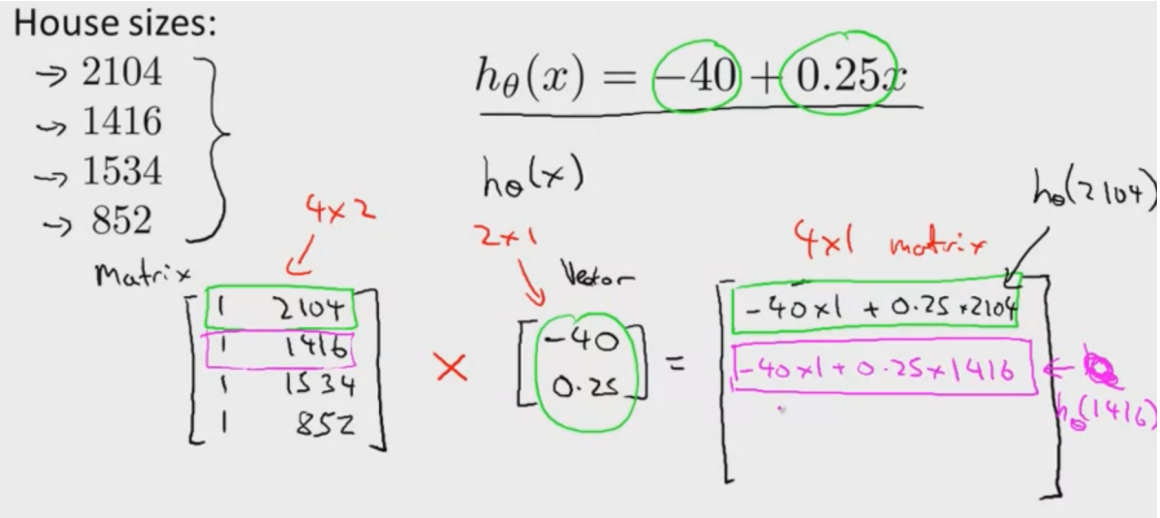
Scalar Multiplication

- Multiply by arbitrary number

Matrix Vector Multiplication



Sum of element wise multiplication for row in Matrix A to column(s) in Matrix B



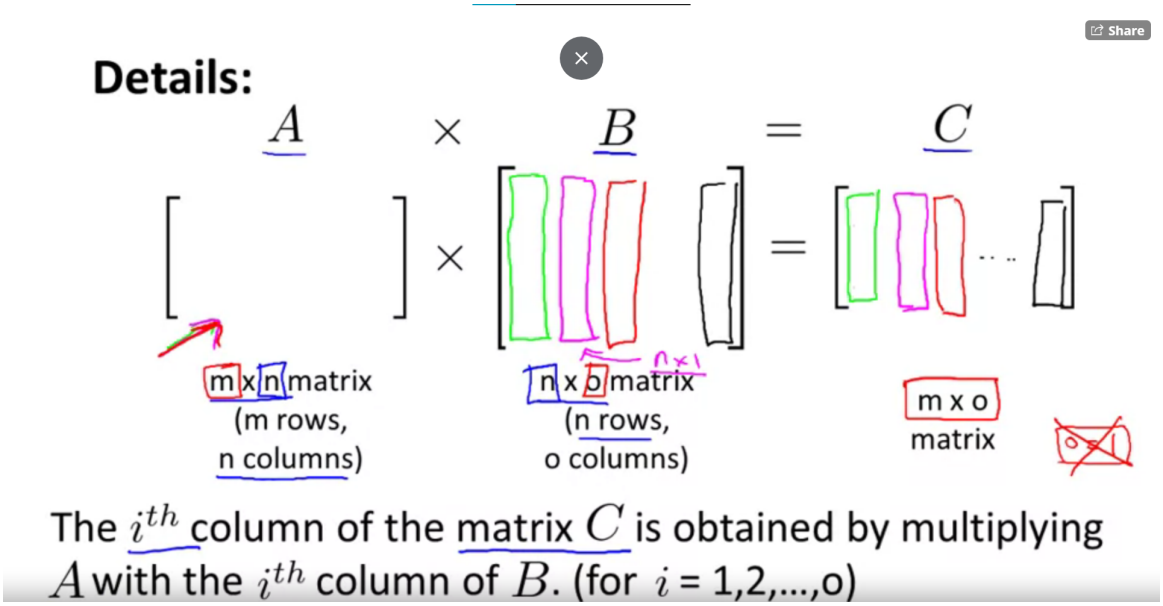
💡 Use matrix and vector to compute linear regression

Linear Regression with Matrices and Vector (m,1)

$$\begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \dots & \dots \\ 1 & x^{(m)} \end{bmatrix} \cdot \begin{bmatrix} b \\ w_1 \end{bmatrix} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \dots \\ \hat{y}^{(m)} \end{bmatrix}$$

Deep Learning with Matrices and Vector (1,m)

$\hat{y} = W \cdot X + b$



House sizes:

$$\begin{Bmatrix} 2104 \\ 1416 \\ 1534 \\ 852 \end{Bmatrix}$$

Have 3 competing hypotheses:

1. $h_\theta(x) = -40 + 0.25x$
2. $h_\theta(x) = 200 + 0.1x$
3. $h_\theta(x) = -150 + 0.4x$

Matrix

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}$$

Matrix

$$\times \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

^ Computing multiple hypothesis at same time

▼ Matrix Multiplication Properties

- Not Commutative Multiplication

  $A \cdot B \neq B \cdot A$

- Associative Multiplication

  $A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$

- Identity Matrix

  $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} A_{3*3} \cdot I_3 = I_3 \cdot A_{3*3}$

▼ Inverse and Transpose

Inverse

- 1 = "identity"
- $3(3^{-1}) = 1$
- Not all Number have inverse : $0(0^{-1}) = \text{undefined}$
- Matrices that don't have an inverse are "singular" or "degenerate" matrices ⇒ Too close to zero

Week 2

▼ Multivariate Linear Regression

▼ Multiple Features

- Feature : x_1, x_2, x_3, x_4
- Number of Feature: n
- Label : y
- $x_2^{(3)}$ = Second Feature of 3rd Training Example

Multivariate Linear Regression

$h_\theta(x) = \theta_0 + \theta_1 \cdot X_1 + \theta_2 \cdot X_{...} \theta_n \cdot X_n$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ ... \\ \theta_n \end{bmatrix} X = \begin{bmatrix} x_0 = 1 \\ x^1 \\ x^2 \\ ... \\ x^n \end{bmatrix}$

$h_\theta(x) = \theta.T \cdot X$

▼ Gradient Descent for Multiple Variables

Hypothesis : $h_\theta(x) = \theta.T \cdot X = \theta_0 \cdot x_0(1) + \theta_1 \cdot x_1 \ldots + \theta_n \cdot x_n$

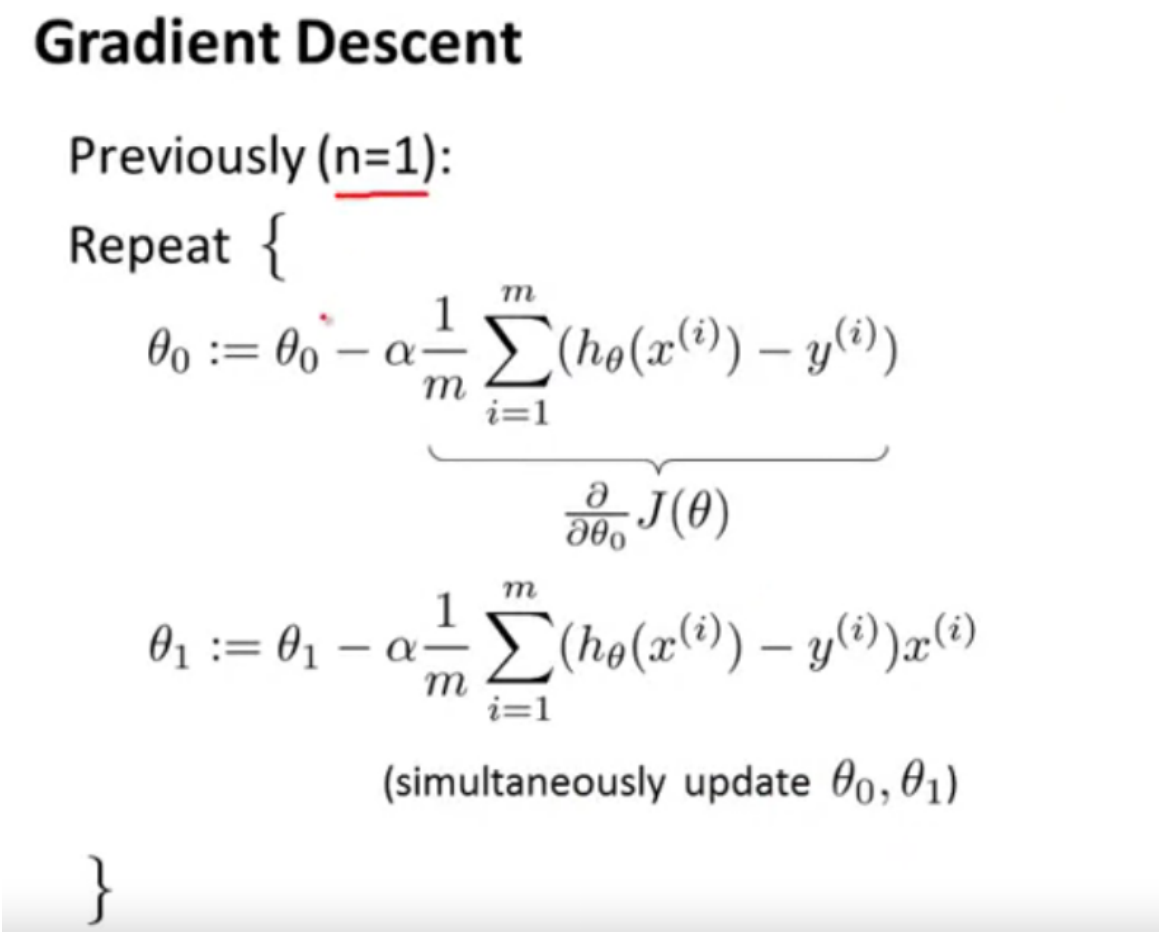
Parameters = θ

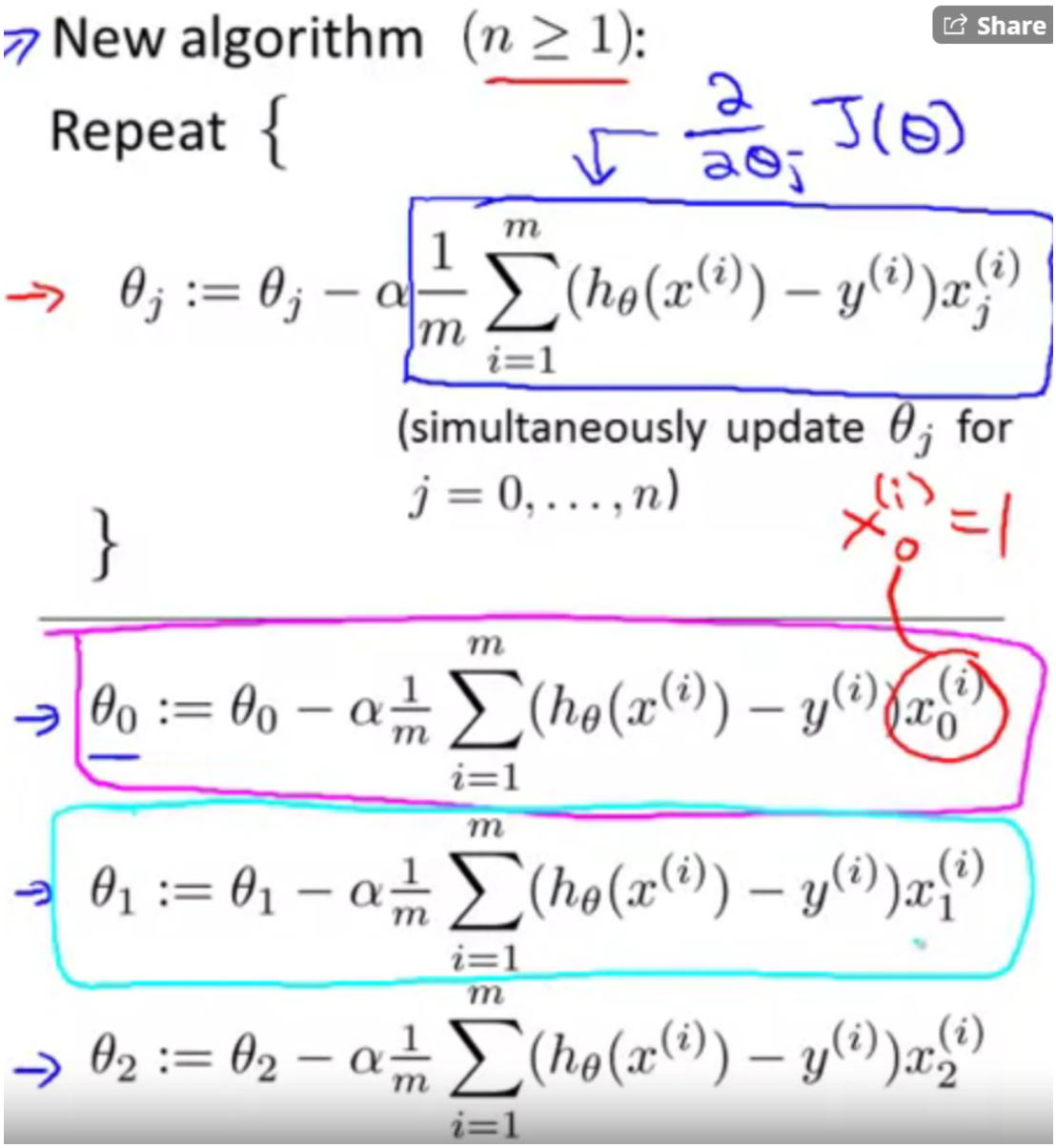
Cost Function : $J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

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

Gradient Descent:

Repeat $\{\theta_j := \theta_j - \alpha * \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$ For j in $n\}$





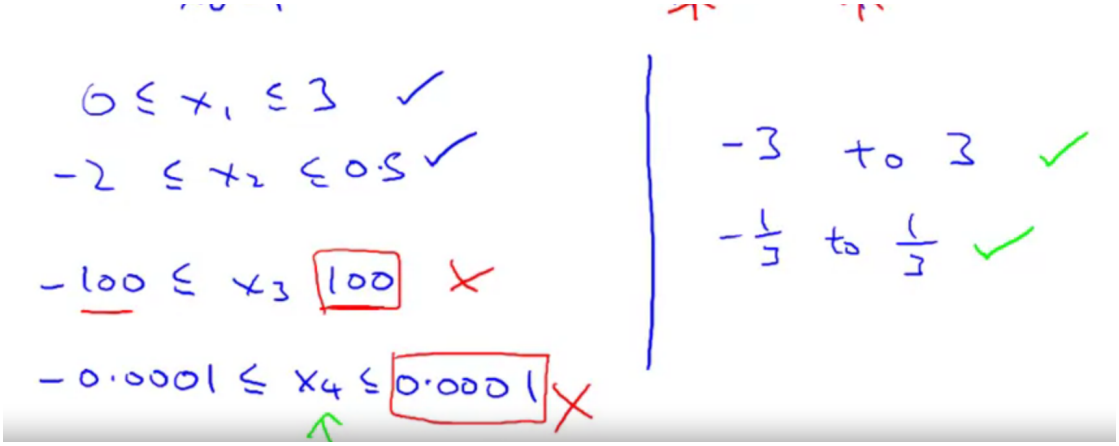
$d\theta_0 = \frac{\partial}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$

$d\theta_1 = \frac{\partial}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - y^{(i)}) x_1^{(i)}$

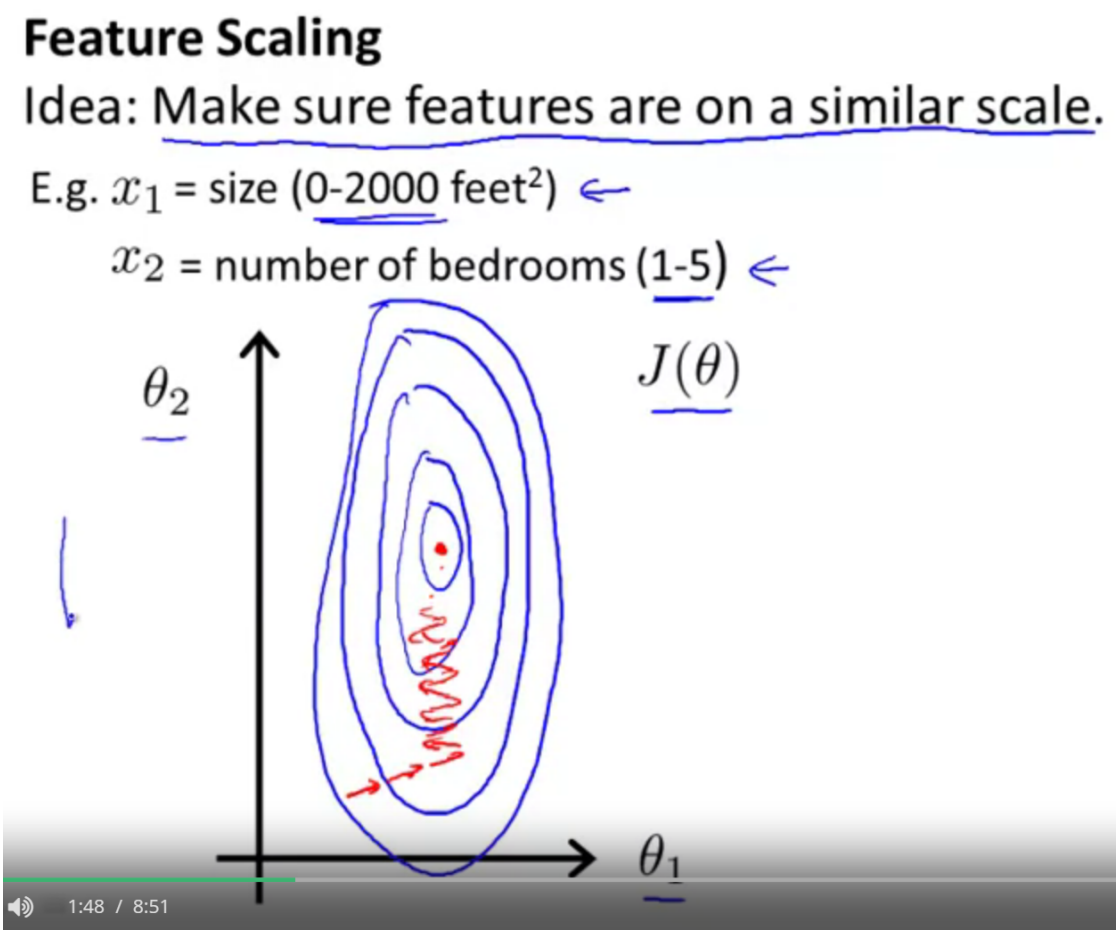
▼ Feature Scaling

Feature Scaling

- Get every feature into approximately $-1 \leq x_i \leq 1$ range

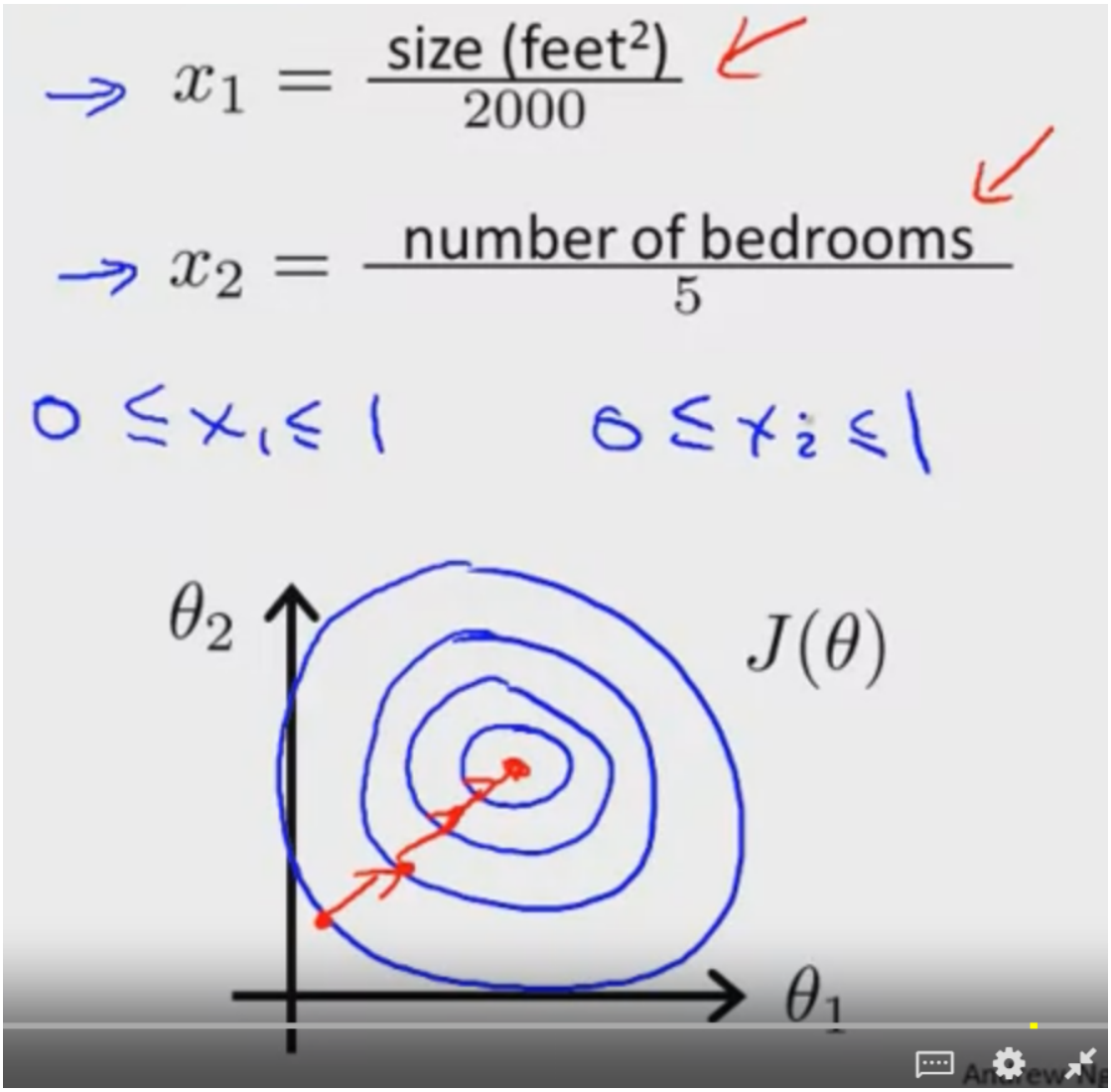


- Scale features to same scale such that contour will approx. to circle than oval



^ longer time for gradient descent to reach global minimum

- Maximum Normalisation



^ Scale by max number of each feature

- Mean Normalization

Mean normalization

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

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

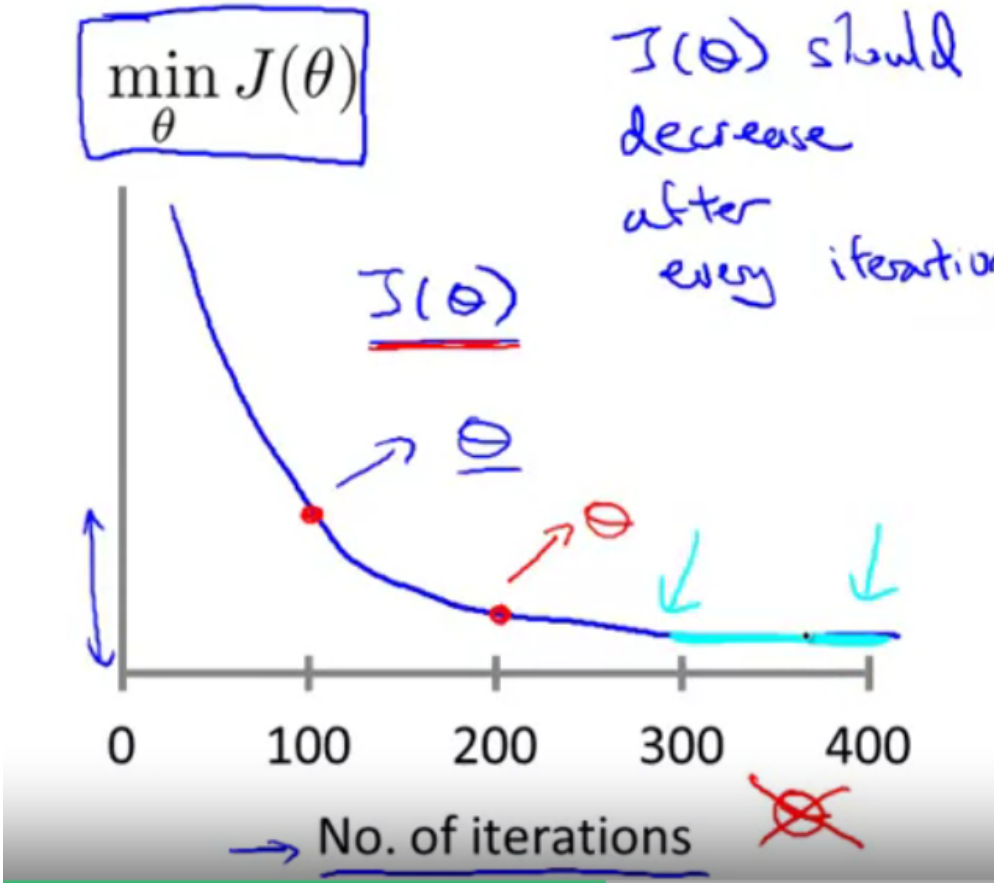
$x_2 = \frac{\#bedrooms - 2}{5}$

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

$$x_1 = \frac{x_1^{(i)} - \mu_1}{(max(x_1) - min(x_1)) \text{ or } \sigma_1}$$

▼ Learning Rate

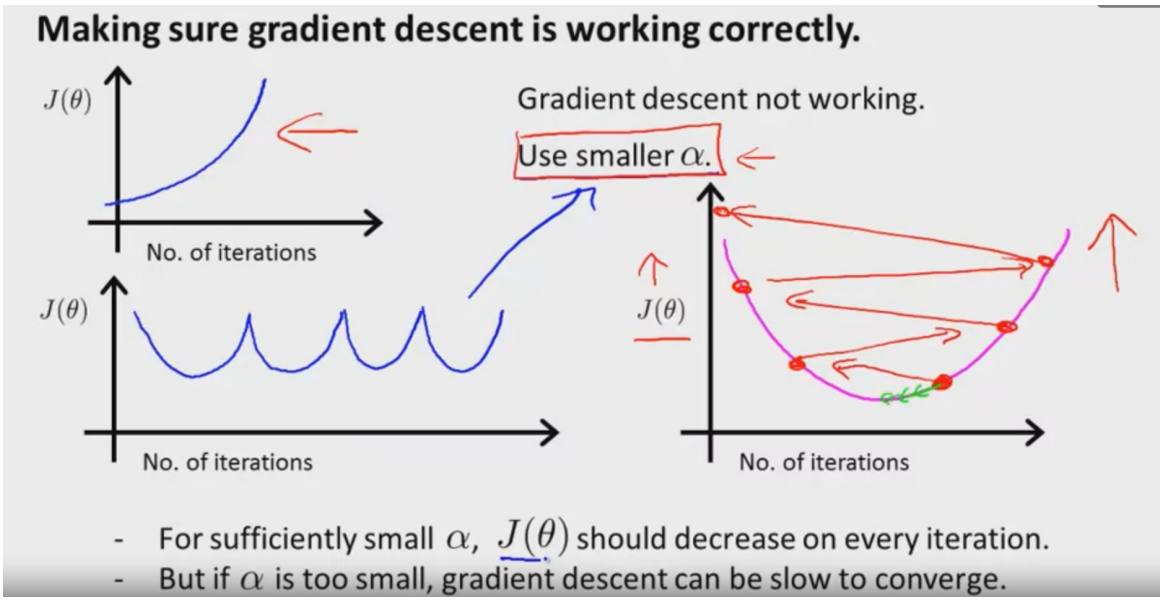
Making sure Gradient Descent Running

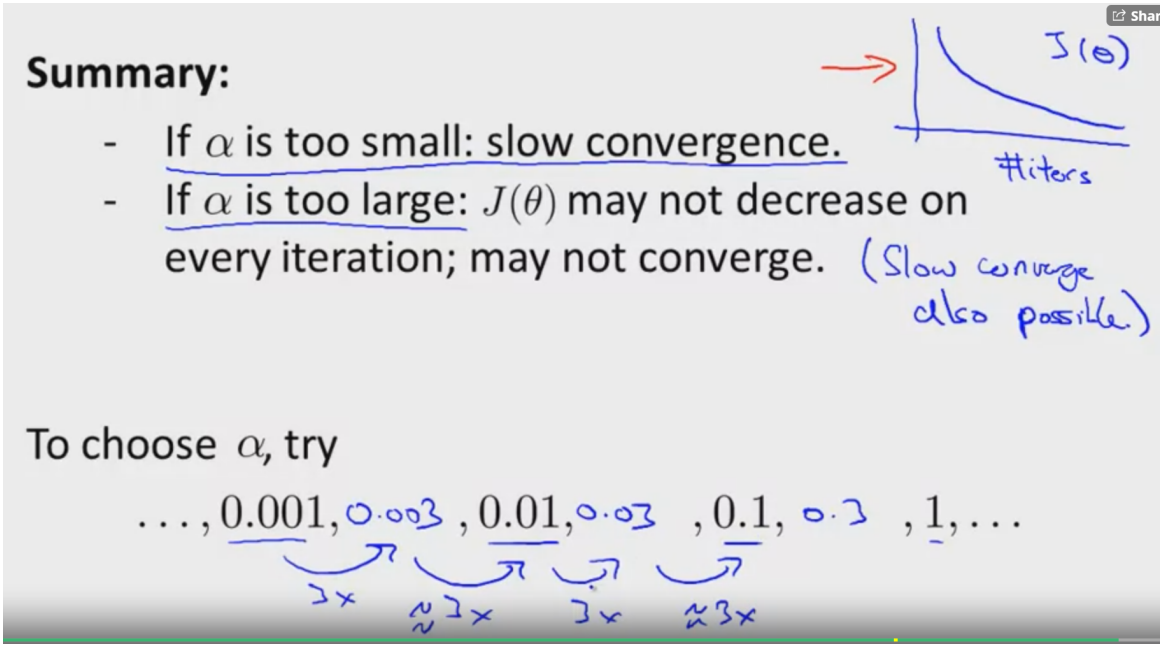


^Plot cost function against number of iterations

^Light blue line shows gradient descent have converged

- Automatic Convergence Test:
  - To tell does gradient descent have converged to global minimum, declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.



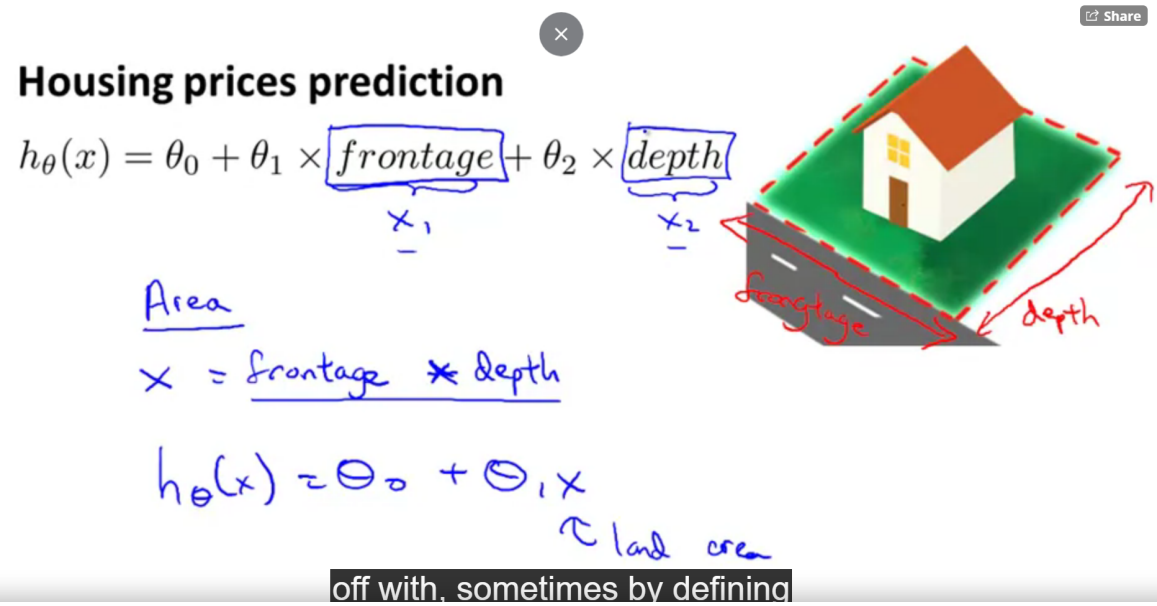


^ 3x fold

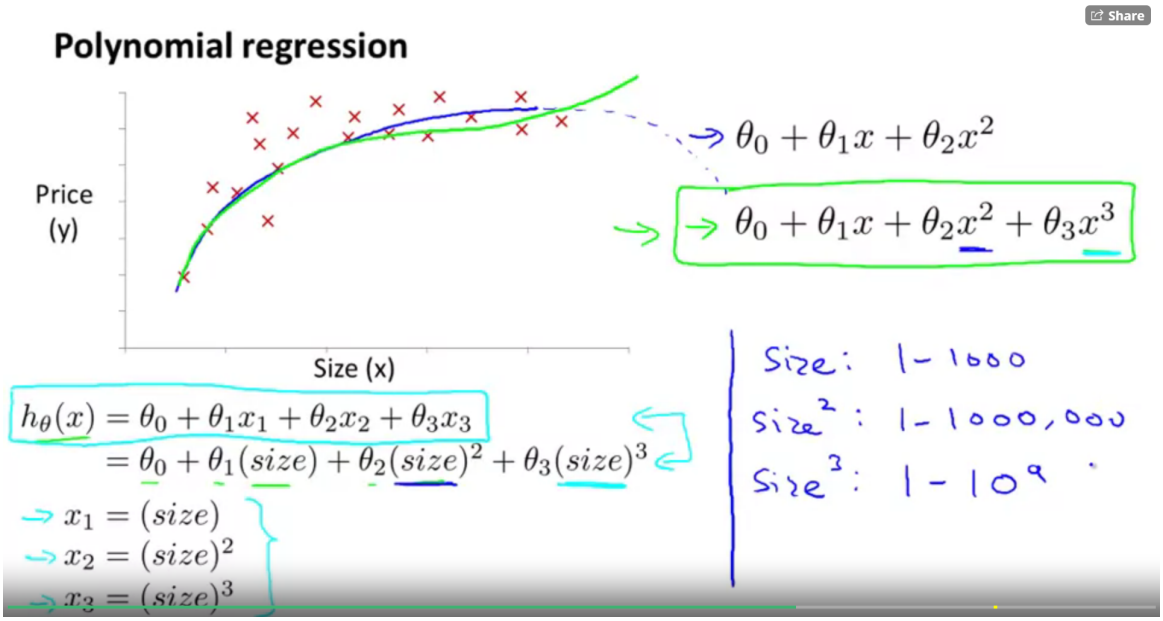
▼ Features and Polynomial Regression

Feature Engineering

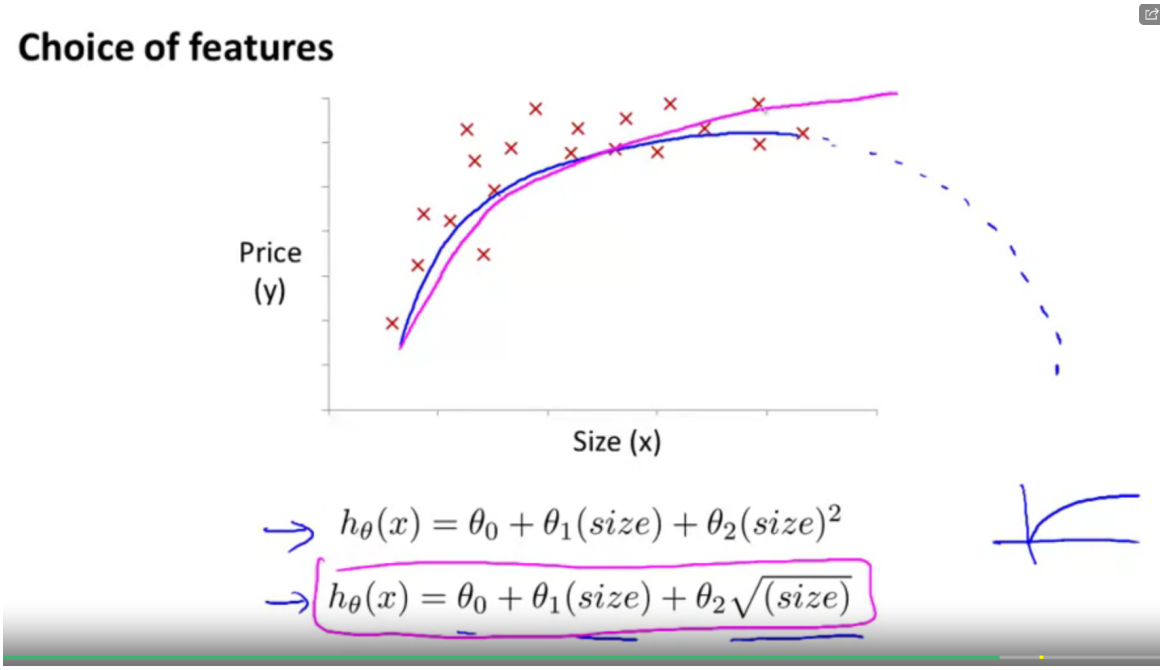
- Generate feature based on insights to the problem



Fit different regression line to the data



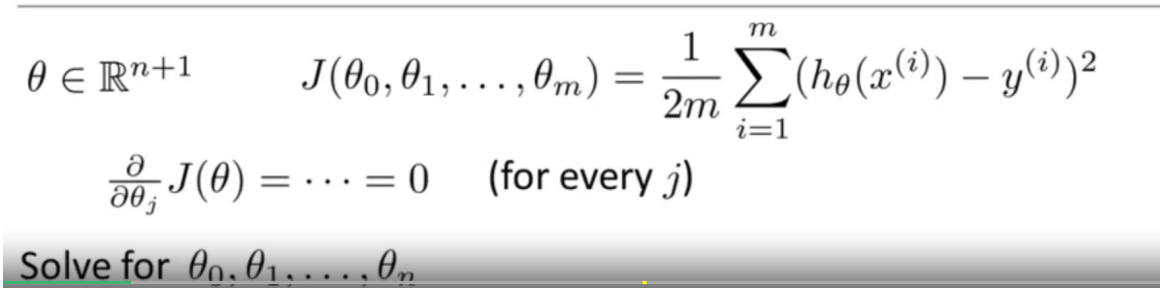
- Quadratic might not be logical as the curve will go down eventually
- Qubic might be a possible choice



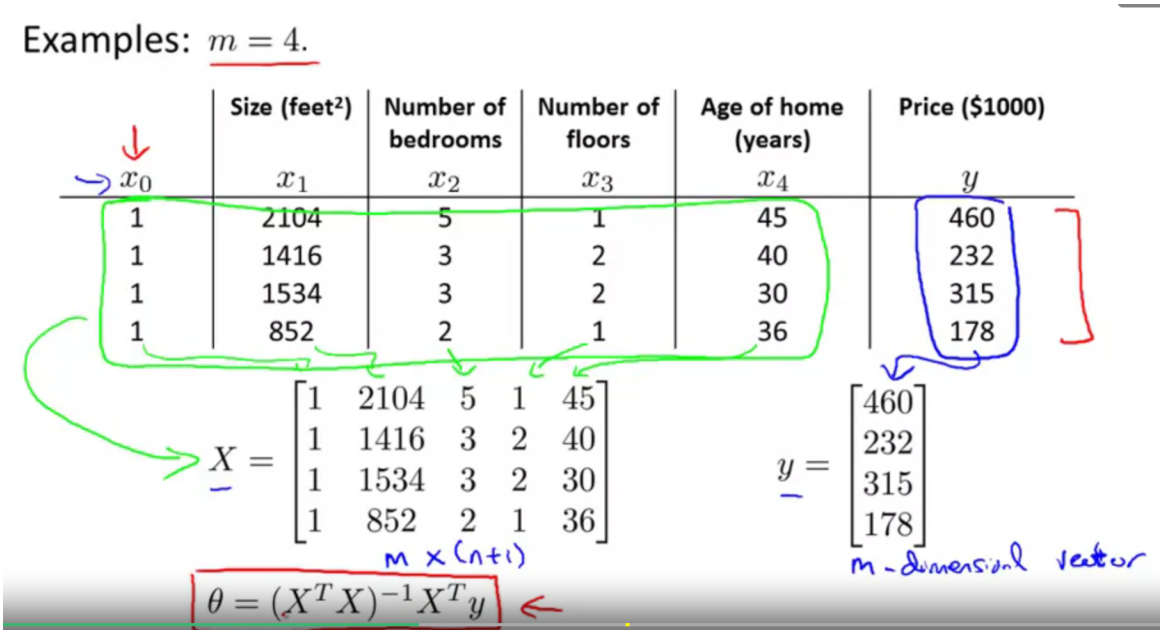
💡 sqrt will retain the "flattening" shape of the graph

▼ Normal Equation

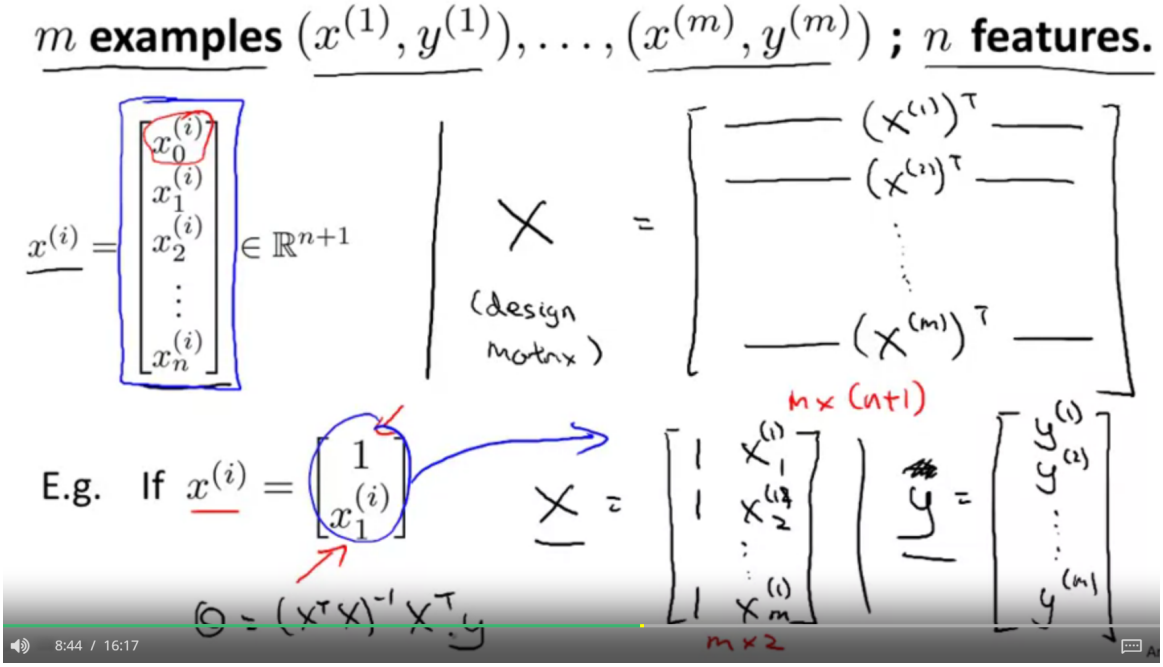
Normal Equation: Method to solve for θ analytically (1 computation only)



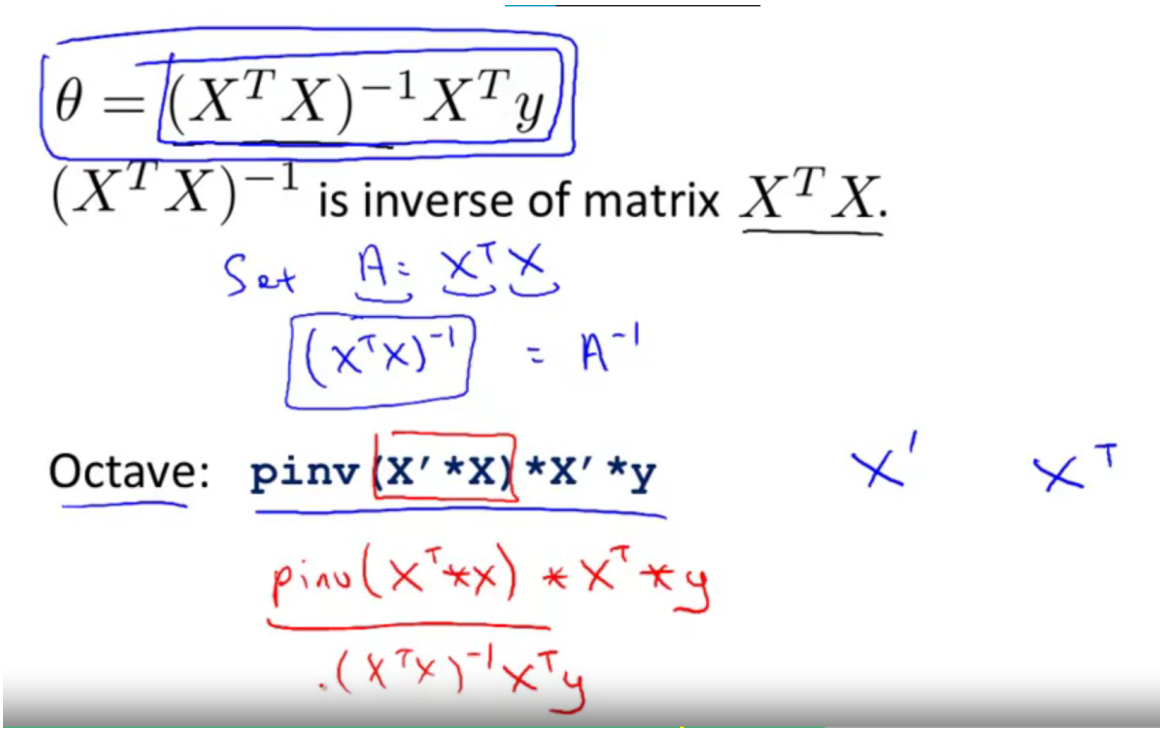
$\frac{\partial}{\partial \theta_j} J(\theta) = 0$ For j in n : Set $\frac{\partial}{\partial \theta_j} J(\theta)/d\theta_j$ to 0 and compute θ that minimize $J(\theta)$



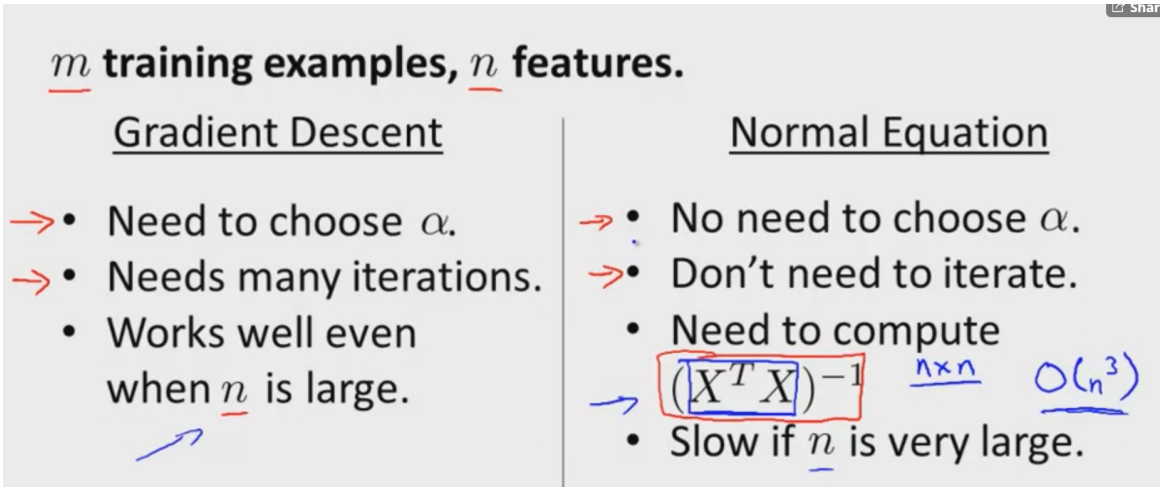
$\frac{\partial}{\partial \theta_j} J(\theta) = \theta = (X.T \cdot X)^{-1} X.T \cdot y$



if (n+1, m) $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$ X(Design Matrix), (m, n+1) $= \begin{bmatrix} 1 & x_1^{(i)} \\ \ldots & \ldots \\ 1 & x_1^{(m)} \end{bmatrix}$



Gradient Descent vs Normal Equation



Normal is slow when n(number of features) is large as computing $(X.T \cdot X)^{-1}$ is slow : $O(n^3)$

Normal Equation for Noninvertibility

Normal equation

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

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

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

- Redundant features (linearly dependent).
  E.g. x_1 = size in feet^2
  x_2 = size in m^2
  $1_m = 3.28$ feet
  $x_1 = (3.28)^2 x_2$
- Too many features (e.g. $m \leq n$).
  $\rightarrow m = 10$
  $\rightarrow n = 100$
  $\theta \in \mathbb{R}^{101}$
  - Delete some features, or use regularization.

Octave : pinv = pseudo-inverse

Week 3

▼ Classification and Representation

▼ Classification (Using Linear Regression Line to Classify)

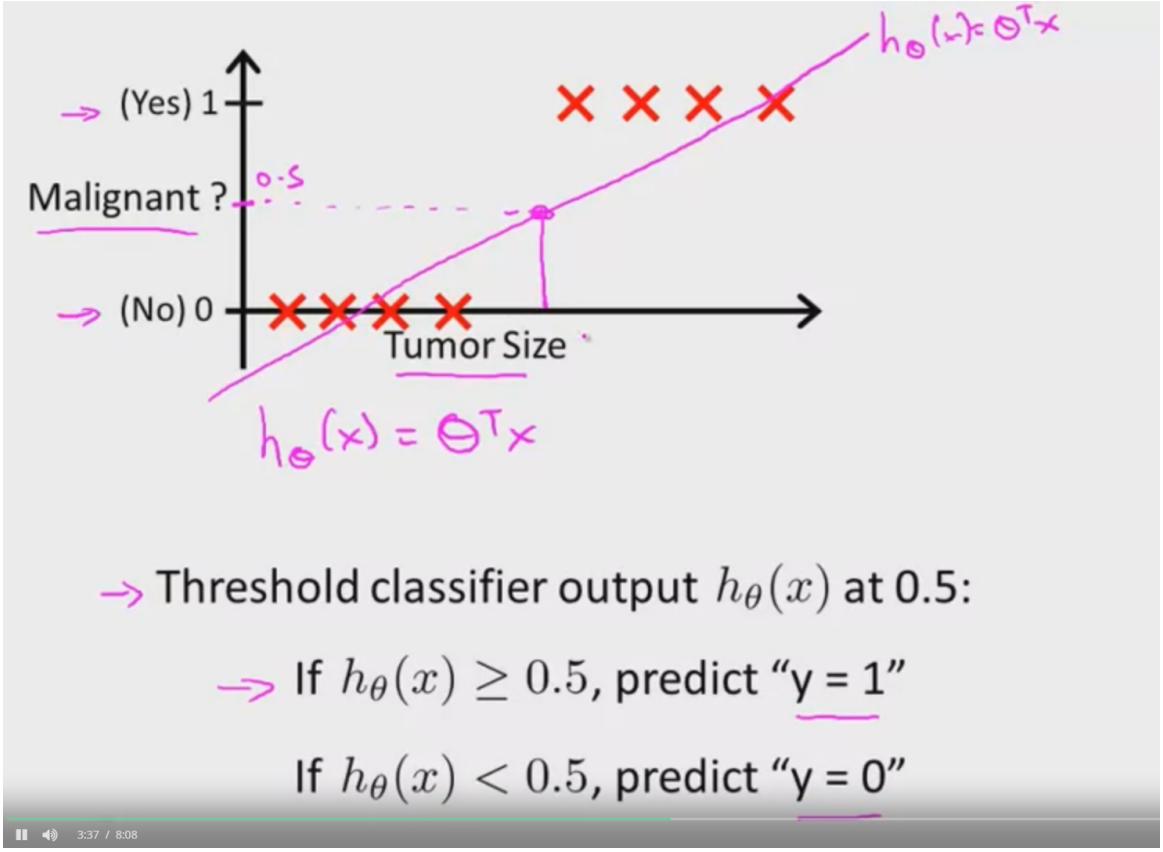
Classification

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

$y \in \{0, 1\}$

0: "Negative Class" (e.g., benign tumor)

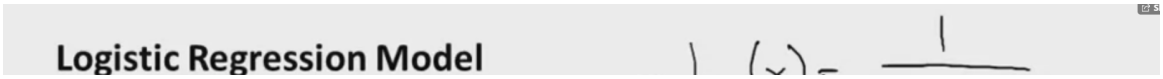
1: "Positive Class" (e.g., malignant tumor)



Outliers will cause great impact to linear regression

▼ Hypothesis Representation (Sigmoid/Activation Func)

$h_\theta(x) = g(\theta.T \cdot X)$
$g(z) = \frac{1}{1+e^{-z}} = \text{Sigmoid}/\ \text{Logistic Function} = 0 \leq g(z) \leq 1$



▼ Decision Boundary

if threshold = 0.5

$h_\theta(x) \geq 0.5 = 1;$
$h_\theta(x) < 0.5 = 0$

From the plot, for $h_\theta(x) \geq 0.5$,

$Z \geq 0$ when $\theta.T \cdot X \geq 0;$
$Z < 0$ when $\theta.T \cdot X < 0$

Decision Boundary is Line Separate both classes given features parameterized by weights(thetas)

$\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0 = positive$

^ Manipulate this inequalities to get an equation line

^ More complex decision boundary with higher order polynomial features

▼ Logistic Regression Model

▼ Cost Function

Optimizing Objectives

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

^ Non-convex function for logistic regression, so cannot be utilized due to gradient descent. because of the non-linear function(sigmoid/logistic), $J(\theta)$ becomes nonconvex function if square cost function is used.

Non-convex: Multiple Local Minimum, Unable to guarantee conversion to Global Minimum

"

^ Cost close to zero if $h_\theta(x)$ is 1 also given y = 1

^ Cost close to zero if $h_\theta(x)$ is 0 also given y = 0

▼ Cost Function and Gradient Descent

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

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

Principle of maximum likelihood estimation

$P(y=1|x;\theta) = h_\theta(x) = \frac{1}{1-e^{-\theta.T \cdot x}}$

Gradient Descent

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

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

^ same $d\theta$ as linear regression

▼ Advanced Optimization of Cost Function

▼ One-vs-all

Multiclass Classification Problem

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

▼ Solving the Problem of Overfitting

▼ Problem of Overfitting

Underfit

- High Bias
  - Higher preconception of something despite data on the contrary
  - e.g. in Linear regression, bias will be the target will increase linearly with feature

Overfit

- High Variance
  - Hypothesis can fit too well to any function, possibly hypothesis is too large, too variable

Addressing overfitting

As not all decision boundary can be visualized(High dimensionality), following ways can be used to address overfitting:

1. Reduce number of features
   - Manually select features to keep
   - Model selection algorithm
2. Regularization
   - Keep all features but reduce magnitude/values of parameters θ_j
   - Works well for lots of features, each contributing a bit to predicting y

▼ Cost Function

Intuition

^ If we want minimize effect of θ_3, θ_4, in my Cost Function, both weight is times with large number such that it Cost will increase if θ_3, θ_4 is big

Regularization

- Smaller value of parameters
- "Simpler" hypothesis
- Less prone to overfitting
- As we don't know which parameter, θ_j to shrink, we do it for all

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

^ j start at 1, by convention θ_0 is not panelized

Role of Regularization Parameter, λ

- Fit training set well
- Keep parameters,θ small that is captured by regularization objective
- λ controls trade off between two goal of fitting well and not overfit

λ cannot be too big, else underfit

if lambda is too large, all parameters are not showing any effect ⇒ underfit

▼ Regularized Linear Regression

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

Regularized Gradient

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

Gradient Descent

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

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

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

}

$\theta_j(1 - \alpha\frac{\lambda}{m})$ have the effect of shrinking θ_j before normal gradient descent update based on cost function

Normal Equation

$$\frac{\partial}{\partial\theta_j}J(\theta) = 0$$
$$\text{such that } J(\theta) \text{ is minimised}$$

$$\theta = (X.T \cdot X + \lambda \begin{bmatrix} 0 & 0 & 0 & ... \\ 0 & 1 & 0 & ... \\ 0 & 0 & 1 & ... \\ ... & ... & ... & 1 \end{bmatrix})^{-1} X.T \cdot y$$
$$\text{eye with first element 0 as } \theta_0 \text{ is not regularized}$$

▼ Regularized Logistic Regression

Cost Function

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

Regularized Gradient

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

Gradient Descent

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

Week 4

▼ Motivations

▼ Non-linear Hypothesis

- Too Many Features for Polynomial Terms (deg = 2)
  - Computational Heavy $O(n^2)$

- Overfit $\frac{n^2}{2}$ Features

^ Given Grayscaled, 50 by 50 Pixel and Polynomial Term, number of features, n = 3 million features (Too much, overfit, time)

▼ Neural Networks

▼ Model Representation I

Single Logistic Regression Unit as Single NN unit

$n_x^{(i)} = s_j$

$w.shape = (n_x, (n_{x-1} + 1[b]))$

> 💡 If layer 1 has 2 input nodes and layer 2 has 4 activation nodes. Dimension of $\Theta^{(1)}$ is going to be 4×3 where $s_j = 2$ and $s_{j+1} = 4$, so $s_{j+1} \times (s_j + 1) = 4 \times 3 sj + 1 \times (sj + 1) = 4 \times 3$.

▼ Model Representation II

Single Value Computation

$$a_j^{(i)} = g(Z_j^{(i)})$$
$$Z_j^{(i)} = \Theta_j^{(i)} \times a_{j-1}^{(i)}$$
$$a := \text{Activation Output}$$
$$g(Z) := \text{Activation Function}$$
$$Z := \text{Perceptron Output of } W \times a_{j-1}$$

Vectorized Implementation

$$\Theta^{(1)} = \begin{bmatrix} \theta_0^{(1)} = b \\ \theta_1^{(1)} \\ \theta_2^{(1)} \\ \dots \\ \theta_{n_x}^{(1)} \end{bmatrix} x = a^{(1)} = \begin{bmatrix} x_0^{(1)} = 1 & x_0^{(2)} = 1 & \dots & x_0^{(m)} = 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)} \\ \dots & \dots & \dots & \dots \\ x_{n_x}^{(1)} & x_{n_x}^{(2)} & \dots & x_{n_x}^{(m)} \end{bmatrix}$$
$$Z^{(2)} = \Theta^{(1)} \cdot a^{(1)}$$
$$a^{(2)} = g(Z^{(2)})$$

> 💡 Weights can learn feature that is useful flexibly without specifically denoting any polynomial terms, enable to learn more complex hypothesis

Architecture: How different neurons are connected to each other

Week 5

▼ Cost Function and Backpropagation

▼ Cost Function

$$L = \text{Total Layer in Network}$$
$$s_l = \text{No. of Layer in } l \text{ unit.}$$
$$K = \text{Number of Classification Classes}$$

Cost Function in Logistic Regression(Binary Classification)

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

Cost Function in Neural Network For Multiclass Classification($K \geq 3$)

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

$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) log(1 - h_\theta(x^{(i)}))_k] + \frac{\lambda}{2m}\sum_{l]1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_1+1}(\Theta_{ji}^{(l)})^2$$

$\Theta_{i0}^{(l)} X_0$: Bias Term, Not Regularized

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

▼ Backpropagation

$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) log(1 - h_\theta(x^{(i)}))_k] + \frac{\lambda}{2m}\sum_{l]1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_1+1}(\Theta_{ji}^{(l)})^2$$

to Minimize $J(\Theta)$, We need to compute partial derivative of each weight(theta) in each layer: $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Computing δ (Error/Gradient) ignoring Regularization

$\delta_j^{(l)}$ = "Error" of Node j in layer l

if $L = 4$:

$\delta^{(4)} = a^{(4)} - y$

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

$g'(z^{(3)}) = a^{(3)} * (1 - a^{(3)})$

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

Backward Propagation With Regularization

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

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

$D_{ij}^{(l)} := \frac{1}{m}(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)})$ if $j \neq 0$ Not Bias

$D_{i0}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}$ if $j = 0$ Dont Regularize Bias

▼ Backpropagation Intuition

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

$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(t)$

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

▼ Backpropagation in Practice

▼ Implementation Note: Unrolling(Flattening) Parameters

$\Theta^{(l)} = \mathbb{R}^{c \times p+1}$

$c = \text{Number of Node in Current Layer}$

$p + 1 = \text{Number of Node in Previous Unit} + 1 \text{ Bias Term}$

Flatten all Matrix into one single Vector before parsing it into optimizing algorithm

▼ Gradient Checking

- Check if gradient descent working even if cost is decreasing

Using a small ϵ to verify the is gradient calculated approximately the same with gradient descent and without gradient descent

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

Gradient Check Matrix

- Flatten Matrix
- Calculate Partial Derivative by adding ϵ towards every θ
- Only for few examples, turn of gradient checking for training

▼ Random Initialization

Zero Initialisation

The node in the same layer will be computing the same function

$a_1^{(2)} = a_2^{(2)}$, Also $\epsilon_1^{(2)} = \epsilon_2^{(2)}$

Random Initialization: Symmetry breaking

- Initialize each $\Theta_{ij}^{(l)}$ to random range of $[-\epsilon, \epsilon]$

▼ Putting it Together

- Pick network architechture
- Reasonable default : 1 hidden layer, or if >1 hidden layer, have same no, if hidden units in every layer

Train Network

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

for i = 1:m:

Forward Prob and Backward Prob

Compute Delta term (+ regularization)

5. Gradient checking to compare backpropagation, then disable it
6. Gradient descent to minimize $J(\Theta)$

Week 6

▼ Evaluating a Learning Algorithm

▼ Evaluating a Hypothesis(Model)

Overfitting of Model

Hypothesis(Model can overfit) and it is hard to visualise with multiple features

Train Test Split : 7/3 Random Sorting Split

Train Θ, calculate loss of test set

▼ Model Selection and Train/Validation/Test Sets

Cross Validation/Validation Set for Model Selection

Test Set for Final Evaluation to Test Model's ability to generalize

> As the probability for model to overfit in BOTH test set and cv set is lower, being more kiasu helps to know the performance of model

▼ Bias(Underfit) vs. Variance(Overfit)

▼ Diagnosing Bias vs. Variance

Train Error Decrease with more robust model(Overfit)

Cross-Validation Error Indicates if Model Underfit($J_{train}(\Theta)$ & $J_{cv}(\Theta)$ high) or Overfit($J_{train}(\Theta)$ high but $J_{test}(\Theta)$ low)

▼ Regularization and Bias/Variance

Too Small or Too High Regularization Hyperparameter, λ can cause overfitting or underfitting.

$J(\theta)$ with regularization term

$J_{cv}(\theta)$ without regularization term

Use J(θ) to Train(Minimize), Use $J_{cv}(\theta)$ to evaluate

▼ Learning Curve

*Plotting the J(θ) againts m (Batch/Training Example Iteration)

Underfitting

- High Error in J_train

Overfitting

- High Gap Between J_train and J_cv

▼ Evaluating a Hypothesis(Revisiting)

- Choose an achitechture, plot the learning curve(Examine overfit/underfit),

▼ Building a Spam Classifier

▼ Prioritizing What to Work On

- Choose and Think of Ways to Generate or Create Features

▼ Error Analysis

> Implement Quick Solution ⇒ Plot Learning Curve(Decide High Bias/High Variance) ⇒ Inspect Wrongly Classified Example(For Classification Problem) ⇒ Develop Appropriate Features and Decision

- Inspect Wrong Example in Test Set and Spot Patterns and Develop Important Features

- Have A Numerical Evaluation Function to Evaluate and make decision

▼ Handling Skewed Data(Imbalanced Class)

▼ Error Metrics for Skewed Classes(Imbalanced Class)

> (P)recision with (P) so deal with False (P)ositive (Actually Negative but predicted to be positive)

Precision

Of all patient we predicted y=1, what fraction actually has cancer?

$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive}+\text{False Positive}}$

Recall

Of all the true positive cases, what fraction did we correctly detect as having cancer?

$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive}+\text{False Negative}}$

▼ Trading Off Precision and Recall

- Setting threshold high ⇒ Predict Less Positive ⇒ Easier Commit False Negative ⇒ Lower Recall, Higher Precision
- Setting threshold low ⇒ Predict More Positive ⇒ Easier Commit False Positive ⇒ Lower Precision, Higher Recall

F1 Score

- Average Does not Work as the lower number is not penalized enough if another number is too high.
- In Example below Algo 3 is the worst Algo but get highest Average Precision Recall
- $\text{F1 Score} = 2\frac{P*R}{P+R}$

▼ Using Large Data Sets

▼ Data For Machine Learning

For certain Cases, getting more data might be helpful for model

If we Assume Given Features is SUFFICIENT to predict y, more data might help

If we Assume Given Features is NOT SUFFICIENT to predict y, more data might not help

Possible Test: Given Feature, can human expert confidently predict y?

Large Data is useful if model tend to overfit(High Variance, Low Bias)

Week 7

▼ Large Margin Classification

▼ Optimization Objective

In Logistic Regression, $h_\theta(x)/z$ have to be very far from 0 to get a lower cost or vice versa

In SVM, $cost_1(z)$ is defined such that if z is greater than _threshold_, cost will be 0 or vice-versa

$cost_1(z)$ and $cost_0(z)$ is used to replace $-log(z)$ and $-log(1-z)$ in SVM than logistic regression

$\frac{1}{m}$ is just a constant and should be remove in SVM (Just diff convention ppl use)

$J(\theta) = A + \lambda B$: Logistic Regression where A is cost, λ is regularization parameter, B is regularization term(Sum of Square of All θ)

$J(\theta) = C * A + B$: SVM where C is regularization parameter, A is cost, B is regularization term(Sum of Square of All θ)

Cost Function of SVM

$$J(\theta) = C\sum_{i=1}^{m}[y^{(i)}cost_1(h_\theta(x^{(i)}) + (1-y^{(i)})cost_0(h_\theta(x^{(i)}))] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

SVM will predict 1 if z ≥0 and predict 0 vice versa (Centered at 0)

Logistic Regression will predict 1 if z ≥ threshold(0.5) and predict 0 vice versa (Range(0,1))

▼ Large Margin Intuition

Due to how Cost Function is construct, we want z to be far from the decision boundary (0)

e.g.

If y =1, we want $\theta.Tx \geq 1$ (Not Just ≥ 0)

If y = 0, we want $\theta.Tx \leq -1$ (Not Just <0)

Penelizing the model by setting large value of C such that to minimize the cost, the model would like $cost_1(z)/cost_2(z)$ to be 0 (e.g. z ≥ 1)

The Intuition of Wanting $h_\theta(x)$ to be more than the threshold value else keep panelizing it creates a margin that is more robust than normal logistic regression(green/purple line)

IF C is very large it is susceptible to outliers and will change the decision boundary accordingly

As $C \approx \frac{1}{\lambda}$, High C means Lower Lambda Means Lower Regularization Means Easier to Overfit, Low C means Higher Lambda Means Higher Regularization Means not Easy to Overfit

▼ Mathematics Behind Large Margin Classification

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

Green = Decision Boundary, Blue = Theta Vector , Green |- Blue

If Bad Decision Boundary, P is small, low margin, panelized

If Good Decision Boundary, P is more, large margin, less panelized

▼ Kernels

▼ Kernels I

From Logistic Regression, To Create a Non-Linear Decision Boundary, we mark use of High Order Polynomial Term to perform computation with the cost of higher computation cost and does not neccessary work.

If SVM, Kernel (e.g. f1, f2, f3) is used as the new features before performing matrices multiplication

Kernel measures the similarity/distance from $x^{(i)}$ to $l^{(i)}$

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

If x is near to $l^{(i)}$, kernel ≈ 1

if x is far to $l^{(i)}$, kernel ≈ 0

From Contour Plot, we observe if σ increase, contour becomes wider without changing magnitude of f_1 when x close to l and vice-versa

By Measuring Distance to certain landmark and multiply with its weight, we can produce a non-linear hypothesis that is based on the location of landmark, l and the weight, θ

▼ Kernels II

Decide Location of Landmark, l

L takes the same location of training example X, $l = X = \mathbb{R}^{m \times n+1}$

$$f = \begin{bmatrix} f_0 = 1 \\ f_1 \\ f_2 \\ ... \\ f_m \end{bmatrix}$$
$$f_i = similarity(x, l^{(i)})$$

Given x, compute $f \in \mathbb{R}^{m+1}$: Predict 1 if $\theta.Tf \geq 0$

Regularization of SVM is done differently for faster computation when m is very large

SVM Hyperparameters

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

High C: Easily Affected by Cost, High Variance, Lower Bias

Low C: Not Easily Affected by Cost, Low Variance, High Bias

σ in Gaussian Kernel

High σ, Graph More Smooth, Effect of Cost Less Drastic, Low Variance, High Bias

Low σ, Graph More Steep, Effect of Cost Higher, High Variance, Low Bias

▼ SVMs in Practice

▼ Using An SVM

Type of Kernel

1. Linear Kernel : predict 1 if $\theta.T \cdot x \geq 0$ /Logistic Regression
   - n large, m small
2. Gaussian Kernel, Choose σ^2 + [FEATURE SCALING]
   - n small, m large
3. Polynomial kernel, Choose Constant, Degree
4. String Kernel, chi-square kernel, histogram intersection kernel

Logistic Regression vs SVMs

n large, m small ⇒ Logistic Regression /SVM (Linear Kernel)

n small, m intermediate ⇒ SVM with Gaussian kernel

n small, m EXTREMELY large ⇒ Add more features and use logistic regression(As it is computational heavy to compute too many features with Gaussian kernel)

Week 8

▼ Clustering

▼ K-means algorithm

K-Means Operation: Cluster Assignment, Move Centroid

1. Randomly assign Cluster Centroids

2. Cluster Assignment: Assign Which data point belongs to which cluster centroids(with shortest distance)

3. Move Centroid: Move Cluster Centroid to the average location of all points

4. Repeat the Process Until The Cluster Centroids does not change much ⇒ K means converged

K-means Algorithm

K = Class, m = Number of Example, c = Assigned Classes

- If a cluster centroid have no points assigned to it, one way is to eliminate the cluster or to randomly assign it to other location

Non-separated clusters: Market Segmentation

In second diagram, unsupervised learning is used as i want to seperate continuous variables into 3 clusters by getting the average of them such that i can design my tshirt size accordingly

▼ Optimization Objective

$c^{(i)}$ Cluster Where i-th example is assigned

μ_k Cluster Centroid if cluster k

$\mu_{c^{(i)}}$ Cluster Centroid of cluster assigned to i-th example

Distortion Cost Function

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

Cluster Assignment step

Minimize J(...) wrt $c^{(1)}, c^{(2)}, ..., c^{(m)}$ holding $\mu_1, \mu_2, ...\mu_k$

Move Centroid Step

Minimize J(...) wrt(With Respect To) $\mu_1, \mu_2, ..., \mu_k$ holding $c^{(1)}, c^{(2)}, ..., c^{(m)}$

▼ Random Initialization

Randomly Initialize $\mu_1, \mu_2, ..., \mu_k \in \mathbb{R}^k$

Random Initialization

- K < m
- Randomly pick K training example
- Set $\mu_1, ..., \mu_k$ equal to these K examples.

Local Optima

- Try Multiple Random Initialization and pick best solution (Lowest Distortion function)
- When k is small, multiple random solution helps better than when k is large as when k is large, it is easier to obtain a good solution

▼ Choosing the Number of Clusters

Elbow method

Find the point where cost function converged ("elbow")

Might be ambiguous if the curve is smooth

Choosing value of K: Downstream Evaluation

- K-means is usually used for later/downstream purpose (e.g. find cluster to design shirt). We Evaluate K-means based on a metric for how well it performs for later purpose.
- E.g. Choose Cluster based on customer evaluation metrics later(e.g. Number Sold, Customer Feedback, etc.)

▼ Motivation for PCA

▼ Motivation I: Data Compression

- Use Less Memory, Speed Up Training Process
- Project 2D data into just 1 Dimension with a line

- Project 3D data into a 2D plane

▼ Motivation II: Visualization

Reduce Dimension into lower dimension for visualization for better intuition of data

▼ Principal Component Analysis

▼ Principal Component Analysis Problem Formulation

Minimize Projection Error : Sum of Square of Error

Reduce from 2-dimenstion to 1-dimension: Find a direction(a vector $u^{(i)} \in \mathbb{R}^n$ to minimize projection error

Reduce n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, ..., u^{(k)}$ to project data while minimizing projection error.

PCA is not linear regression

Linear regression minimize vertical(y) distance while PCA minimize shortest distance to line

Linear Regression minimize error respect to a variable y while PCA minimize error wrt All Variable

▼ Principal Component Analysis Algorithm

Data Preprocessing: Feature Scaling/Mean Normalization

$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j / max(x_j)}$

Principal Component Analysis (PCA) Algorithm

- Compute k-dimension vectors $u^{(1)}, u^{(2)}, ..., u^{(k)}$
- Compute n-dimension into k-dimension $\begin{bmatrix} z_1 \\ z_2 \\ ... \\ z_m \end{bmatrix}$

Single Value Decomposition

GET u matrix, with svd(sigma) where sigma = $\frac{1}{m} \sum_{i=1}^{m} (x^{(i)} \cdot x^{(i)}.T)$

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

U_reduce = U[:, 1:k)

z = Ureduce.T · x;

▼ Applying PCA

▼ Reconstruction from Compressed Representation

Reversing the process and get approx of Higher Dimension Data with Reduced Data and Vector

▼ Choosing The Number of Principal Component

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 data

  $\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2$

- Choose k to be smallest value so that

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

$$\text{"}99.9\%\text{ of variance is retained"}$$

Choosing k Algo

For k in range(1,m+1):

Compute $U_{reduce}, z^{(1)}, z^{(2)}, ..., z^{(m)}, x_{approx}^{(1)}, ..., x_{approx}^{(m)}$

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

~OR~

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

$s=I(n)$

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

▼ Advice of Applying PCA

Supervised Learning Speedup

$x^{(i)}=\mathbb{R}^{10000}$: 10000 Features in single training example

Extract inputs:

Unlabeled dataset: $x^{(1)}, x^{(2)}, ..., x^{(m)}$

Apply PCA and obtain $z^{(1)}, z^{(2)}, ..., z^{(m)}$

New Training Set:

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

Bad Use of PCA: Prevent Overfitting

- As PCA will reduce some valuable information, better to prevent overfit with regularization term than PCA

PCA is Used Sometimes but not every time

- PCA is not default but an option to reduce the load and memory consumption of model

Week 9

▼ Density Estimation

▼ Problem Motivation

Given Dataset Find Is X_{test} anomalous (Unusual?)

Construct Model p(x) and if p(X_test) < α → Flag anomaly

> Contour Shows probability of getting such data with contour probability peak in center

▼ Gaussian Distribution

Parameter Estimation

Estimate μ: Population Mean, σ^2: Population Variance

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

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

▼ Algorithm

Assume Each Feature is Normally Distributed and Independence between variables,

Find Probability of Getting Each Feature from their respective normal distribution

$$p(x) = \Pi_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$
$$\Pi = \text{Product of Each Term}$$

Anomaly detection algorithm

1. Choose Features x_j

2. Fit parameters $\mu_1, ..., \mu_n, \sigma_1^2, ..., \sigma_n^2$
3. Given new example x, compute $p(x)$

   Anomaly if $p(x) < \epsilon$

▼ Building an Anomaly Detection System

▼ Developing and Evaluating an Anomaly Detection System

Importance of real-number evaluation

- Having a evaluation number to make decision
- Assume we have some labeled data: (y=0 normal, y=1 anomalous)
- Training Set (Assuming NO ANOMALOUS ⇒ To calculate μ, σ^2)
- CV & Test Set to have Anomalous data slipped in

Algorithm Evaluation

1. Fit model $p(x)$
2. On Cross validation set, predict the label y
3. Use Evaluation Metrics to evaluate or chose ϵ:
   - Confusion Matrix
   - Precision/Recall
   - F1 Score

▼ Anomaly Detection vs Supervised Learning

Anomaly Detection

- Extremely Small number of Positive Cases (1-20) : Hard to learned by model

Supervised Learning

- Large number of positive and negative examples

▼ Choosing What Features to Use

Plot Data to Visualize the distribution

Transform data if not normally distributed

Error Analysis for anomaly detection

- Plot out Misclassified Data and Come out with new features to identify the anomalous
- x2 = New Feature

▼ Multivariate Gaussian Distribution

▼ Multivariate Gaussian Distribution

If We only calculate gaussian based on individual features, the relationship between features tend to be ignored

Multivariate Gaussian (Normal) Distribution

Model p(x) in one go

Parameter: μ, Σ

▼ Anomaly Detection using the Multivariate Gaussian Distribution

Parameterized by $\mu \in \mathbb{R}^k, \Sigma \in \mathbb{R}^{(n \times n)}$

▼ Predicting Movie Ratings (Recommender System)

▼ Problem Formulation

Given r(i,j) and y(i,j) predict missing value of ?

▼ Content Based Recommendations

Each Movie Have Features x1,x2 ...

Treat as Multiple Linear Regression Model where for Each User j, predict rating for movie i with $(\theta^{(j)}).T \cdot x^{(i)}$ stars

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

Computing Cost for All Users (Similar to Neural Network)

▼ Collaborative Filtering

▼ Collaborative Filtering

Feature Learning:

Assume we have $\theta^{(j)}$ that tells how much a user like certain features, we predict the Feature score of each movie based on user rating

Find $x^{(i)}$ such that $\left(\theta^{(i)}\right)^T \cdot x^{(i)} \approx Rating_j$

Chicken and Egg problem where randomly initialize theta and guess feature and further optimize theta and further optimize features

▼ Collaborative Filtering Algorithm

Treat Cost Function parameterized by features, x and user preference θ

Collaborative Filtering algorithm

1. Initialize x, theta to small random values
2. Minimize J(x, theta) and perform gradient descent (Take Note no More Bias Term for x and theta)

▼ Low Rank Matrix Factorization

▼ Vectorization: Low Rank Matrix Factorization

Fine Related Movies

▼ Implementational Detail: Mean Normalization

When User Rate None Movie, With Current Cost Function, we will predict user theta to be 0 and predict 0 as their rating

We could perform Mean Normalization such that if theta is 0, we can still fill in mean value for each movie of that user

Week 10

▼ Gradient Descent with Large Datasets

▼ Learning With Large Datasets

Use Low Bias Model and train with a lot of data

It takes 100 Million Example just to perform one step of gradient descent is very computational heavy

Use a subset of training example and Verify model is having High Variance Problem before using high number of training sets

▼ Stochastic Gradient Descent

Computation heavy to compute derivative term of cost when m is large [Batch Gradient Descent]

Stochastic Gradient Descent: Gradient Descent for Every Example

1. Randomly Shuffle Training Dataset
2. Repeat: [Epoch]

   For i = 1, ... , m:

   Compute Gradient of Cost for i-th training example

Stochastic Gradient Descent Could not find optimum path towards global minimum as it is highly volatile by a single example

▼ Mini-Batch Gradient Descent

Batch gradient descent: Use all m example in each iteration

Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

Could Perform Vectorization for parallel computation

b in range (2,100) <approx>

▼ Stochastic Gradient Decent Convergence

Make Sure Convergence

Batch Gradient Descent:

Plot J_train(\theta) for every iteration with m example

Stochastic Gradient Descent:

Compute Cost of Each i-th example before gradient descent

Plot out Average Cost for every (Number) Iteration

Use a Decreasing Learning Rate

- Model can converge and descent to global minimum better but at the cost of having two extra parameter

▼ Advanced Topics (SGD and Map Reduce)

▼ Online Learning

Let model to learn from stream of data online

Get Example, Update theta using that example

▼ Map Reduce and Data Parallelism

Map Reduce: Split Computation of Cost to Multiple Machine before combining them and perform gradient descent

Map Reduce: To distribute workload in for summing term into multiple machines

Week 11

▼ Photo OCR(Optical character Recognition)

▼ Problem Description and Pipeline

1. Text Detection
2. Character Segmentation
3. Character classification

Pipeline

▼ Sliding Windows

Supervised Learning for fixed size detection

Slide Through Each Stride and produce prediction

Text Detection

Run Same Sliding Window And Parse Through Expansion Algorithm before verifying aspect ratio to identify potential text

Text Segmentation

Sliding Window to identify split between characters

▼ Getting Lots of Data and Artificial Data (Data Augmentation)

Text Classification Problem:

Use font online and synthesis own data by pasting font into random background

Data Augmentation with distortions

▼ Ceiling Analysis: What Part of Pipeline to Work on Next

Giving Correct Prediction for one part of pipeline and analyse which part of pipeline should be spent more time with