Week 1 What is Machine Learning

Introduction

What's ML

- Early: Arthur Samuel
- Late: Tom Mitchell

Example: playing checkers

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

(task - T, experience - E, performance - P)

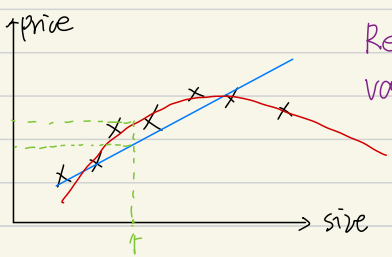
Supervised / Unsupervised

Supervised Learning

监督学习

- "right answers" given, data set given
- e.g. House price prediction

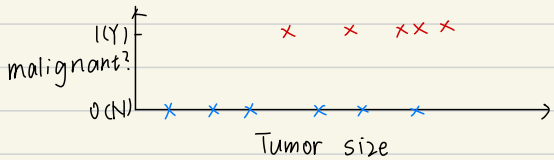
Regression



Regression: Predict continuous valued output

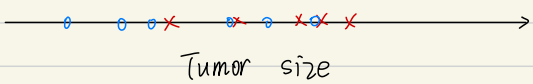
classification

- e.g. Breast cancer (malignant, benign)

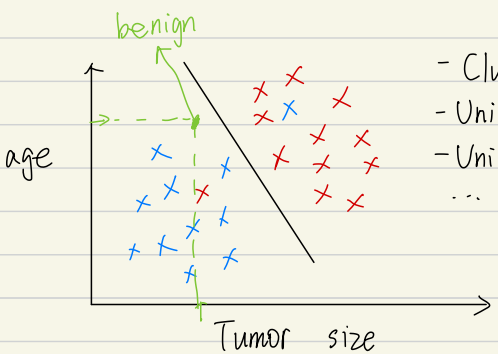


Classification: Descrete valued output (0 or 1)
(can also have more than 2 values)

Another way to plot:

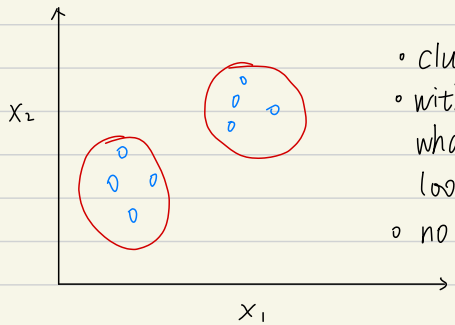


with many features



- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape
- ...

Unsupervised Learning
无监督学习

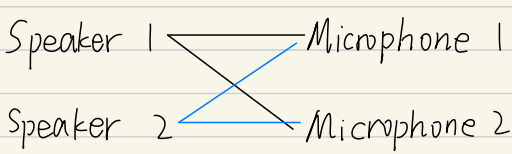


- clustering algorithm
- with little or no idea what the result should look like.
- no feedback for prediction

e.g. google news , genes

- organize computing clusters
- social network analysis
- market segementation
- astronomical data analysis

Cocktail party problem:



Octave

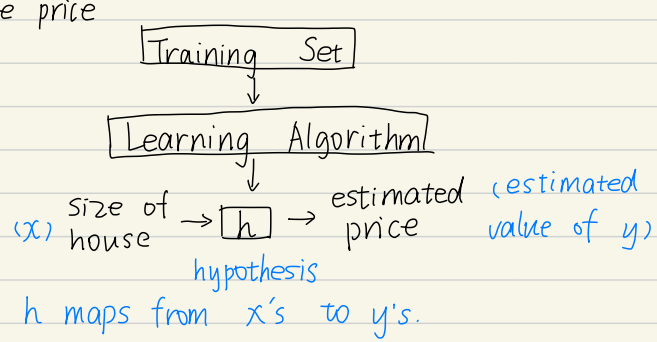
```
[W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

Model and Cost Function

Model Representation

Notation: m = number of training examples
x's = "input" variable / features
y's = "output" variable / "target" variable
(x, y) = one training example
$(x^{(i)}, y^{(i)})$ = i^{th} training example

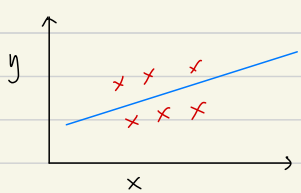
e.g. house price



how to represent h?

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

shorthand: $h(x)$



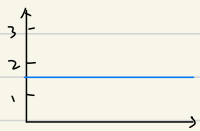
- Linear regression with one variable
- Univariate linear regression.
  - univariate: one variable

Cost Function

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

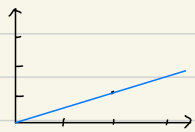
θ_i's : parameters

- How to choose θ_i's ?



$\theta_0 = 1.5$

$\theta_1 = 0$



$\theta_0 = 0$

$\theta_1 = 0.5$

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

- $\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$ why $\frac{1}{2}$?
  convenience for the gradient descent.

cost function

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

also called : squared error function
mean squared error

- find θ_0, θ_1 to minimize $J(\theta_0, \theta_1)$

Cost Function Intuition I

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

Parameters: θ_0, θ_1

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

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

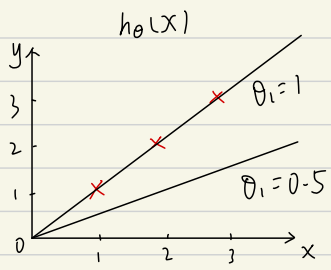
- Simplified: $h_\theta(x) = \theta_1 x$ (Let $\theta_0 = 0$)

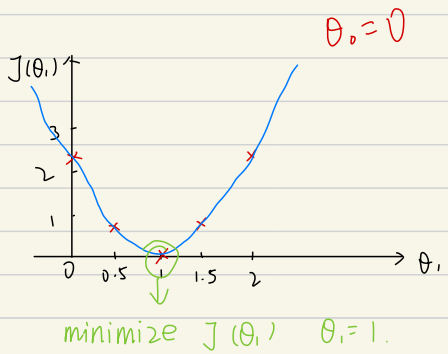
  $J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

  $\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$

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

$J(\theta_1)$: function of the parameter θ_1



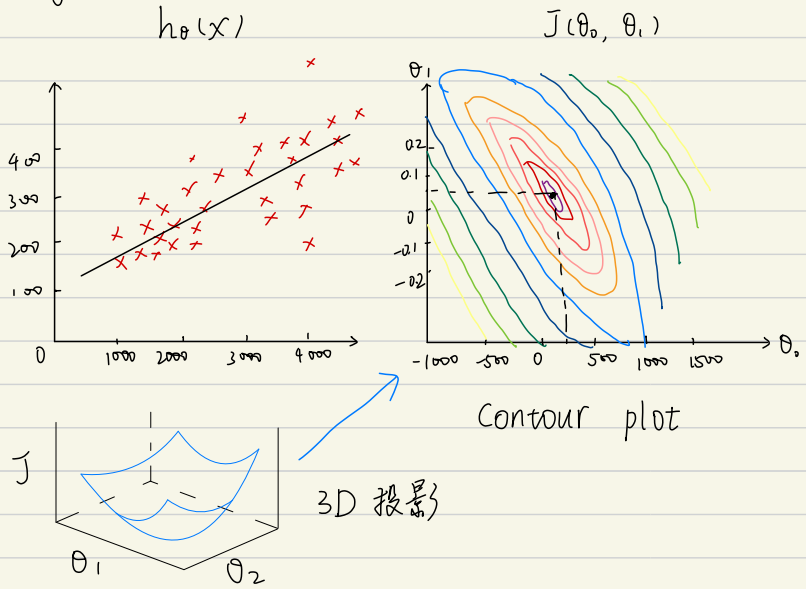


Cost Function Intuition II

- Contour plot: a graph that contains many contour lines.
- Contour line: (of a two variable function) has a constant value at all points of the same line.

Contour plot

- e.g.



Parameter Learning

Gradient Descent

Have some function $J(\theta_0, \theta_1)$ $\quad J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$ $\quad \min_{\theta_0 \cdots \theta_n} J(\theta_0, \theta_1, \ldots, \theta_n)$

Outline:
- start with some θ_0, θ_1 (say $\theta_0 = 0$, $\theta_1 = 0$)
- keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum

Gradient descent algorithm

repeat until convergence {

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

}

α: learning rate (the size of each step)

Note:
- Assignment $a := b$
- Truth assertion $a = b$

$a = a + 1$ ×

- Correct: Simultaneous update

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

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

$\theta_0 := temp0$

$\theta_1 := temp1$

- Incorrect

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

$\theta_0 := temp0$

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

$\theta_1 := temp1$

Question: Why simultaneous update?

Gradient descent Intuition

Gradient descent algorithm:

repeat until convergence {

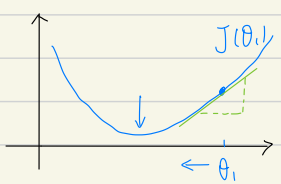
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ (simultaneously update $j = 0$ and $j = 1$)

}

learning rate

derivative

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



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

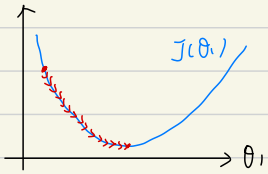
≥ 0

$\theta_1 := \theta_1 - \alpha \cdot$ (positive number)

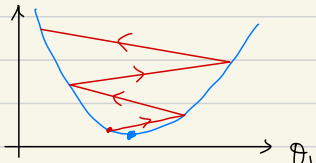
$\frac{d}{d\theta_1} J(\theta_1) \leq 0$

$\theta_1 := \theta_1 - \alpha$ (negtive number)

- adjust α → converge in a reasonable time
- if α too small, gradient descent can be slow
- if α too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



α too small



α too large

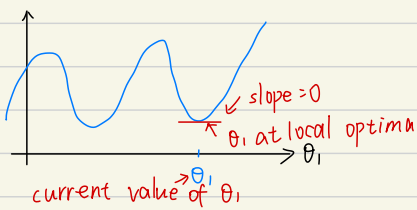
- How does gradient descent converge with a fixed step size α?

  approach the bottom of the convex function

  $\rightarrow \frac{d}{d\theta_1} J(\theta_1)$ approaches 0

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

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



$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{d}{d\theta_1} J(\theta_1)}_{=0}$$

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

$\theta_1 := \theta_1$

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

Gradient descent for linear regression

- Gradient descent algorithm:

  repeat until convergence {

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

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

  }

- Linear regression Model:

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

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

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

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

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

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

- Gradient descent algorithm:

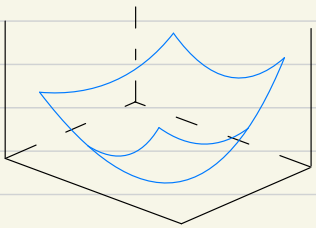
  repeat until convergence {

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

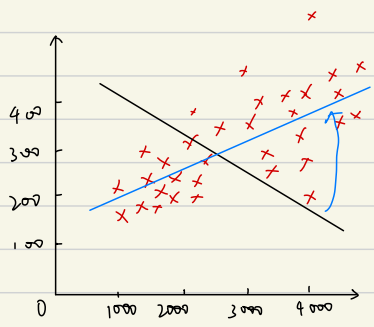
  $\theta_1 := \theta_1 - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}}$ ($\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$)

  } update θ_0 and θ_1 simultaneously

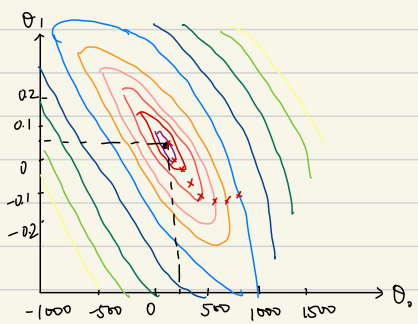
- Convex function: Bowl-shaped



$h_\theta(x)$



$J(\theta_0, \theta_1)$



"Batch" Gradient Descent

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

Linear Algebra Review

Matrices and Vectors

- Matrix: Rectangular array of numbers (2-dimensional arrays)

$$\begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

4 × 2 matrix — $\mathbb{R}^{4\times 2}$

2×3 matrix — $\mathbb{R}^{2\times 3}$

- Dimension of matrix: number of rows × number of cols

- Matrix Elements (entries of matrix)

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

A_{ij} = "i, j entry" in the i^{th} row, j^{th} column.

$A_{11} = 1402$ $A_{12} = 191$ $A_{32} = 1437$ $A_{41} = 147$

~~A_{43}~~ = undefined (error)

- Vector: An $n \times 1$ matrix

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$ ← 4-dimensional vector $\mathbb{R}^4$

$y_i = i^{th}$ element $y_1 = 460$, $y_2 = 232$, ...

1-indexed vs 0-indexed

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

(math) (machine learning)

- A, B, C, X — matrix (uppercase)
  a, b, x, y — number, scalar, vector (lowercase)

Addition and Scalar Multiplication

- Matrix Addition

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0.5 \\ 4 & 10 \\ 3 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \end{bmatrix} = \text{error}$$

- Scalar Multiplication

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} \times 3$$

$$\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} / 4 = \frac{1}{4} \begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{3}{2} & \frac{3}{4} \end{bmatrix}$$

- Combination of Operands

$$3 \times \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix} / 3$$

scalar multiplication

scalar division

Matrix Vector Multiplication

Example: $\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$

Details:

$$A \times x = y$$

$$\begin{bmatrix} \quad \end{bmatrix} \times \begin{bmatrix} \quad \end{bmatrix} = \begin{bmatrix} \quad \end{bmatrix}$$

$m \times n$ matrix (m rows, n columns)

$n \times 1$ matrix (n-dimesional vector)

m-dimesional vector

To get y_i, multiply A's i^{th} row with elements of vector x, and add them up

- House sizes:

2104
1416
1534
852

$$h_\theta(x) = -40 + 0.25x$$

Matrix Matrix Multiplication

matrix: (4×2)   Vector: (2×1)

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 \\ 0.25 \end{bmatrix} = \begin{bmatrix} -40 \times 1 + 0.25 \times 2104 \\ -40 \times 1 + 0.25 \times 1416 \\ -40 \times 1 + 0.25 \times 1534 \\ -40 \times 1 + 0.25 \times 852 \end{bmatrix}$$

Prediction = DataMatrix × Parameters

(Simplify the code, more efficient)

for i = 1 : 4,
    prediction(i) = ...

Example

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 11 \\ 9 \end{bmatrix} \qquad \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

Details: A × B = C

$[\quad]$ $[\,[\,]\,[\,]\,[\,]\cdots[\,]\,]$ $[\,[\,]\,[\,]\,[\,]\cdots[\,]\,]$

$m \times n$ $n \times o$ $m \times o$

House sizes:
2104
1416
1534
852

Have 3 competing hypotheses:

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

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

Matrix Multiplication Properties

- $3 \times 5 = 5 \times 3$ "Commutative"
- Let A and B be matrices. Then in general, $A \times B \neq B \times A$. (not commutative)

- $3 \times 5 \times 2 = (3 \times 5) \times 2 = 3 \times (5 \times 2)$ "Associative"
- $A \times B \times C = (A \times B) \times C = A \times (B \times C)$

- Identity Matrix. Denoted I (or $I_{n \times n}$)

I is identity. $I \times z = z \times I = z$ for any z.

Examples of identity matrices

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

Inverse and Transpose

For any matrix A, $A \cdot I = I \cdot A = A$

($m \times n$) ($n \times n$) ($m \times m$) ($m \times n$) ($m \times n$)

1 = "identity" $3 \times (3^{-1}) = 1$ ($\frac{1}{3}$) $12 \times (12^{-1}) = 1$ ($\frac{1}{12}$) $0 \cdot (0^{-1}) = 1$ undefined

- Matrix inverse: if A is an $m \times m$ matrix (square matrix (# rows = # columns)), and if it has an inverse, $AA^{-1} = A^{-1}A = I$

  e.g. $\begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{2 \times 2}$

  code: pinv(A)

- Matrices that don't have an inverse are "singular" or "degenerate"

- Matrix Transpose

  Example: $A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}$ $A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$

  Let A be an $m \times n$ matrix, and let $B = A^T$. Then B is an $n \times m$ matrix, and $B_{ij} = A_{ji}$

Week 2 Multiple Features

Multivariate Linear Regression

Multiple Features

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

m

Notation: m = number of training examples $n = 4$

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example

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

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix} \in \mathbb{R}^4 \qquad x_3^{(2)} = 2$$

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

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

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

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

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

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

Gradient Descent for Multiple Variables

- Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
- Parameters: θ $(n+1)$-dimesional vector
- Cost function: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
- Gradient descent:

  Repeat until convergence{

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

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

- $n=1$: Repeat{

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

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

  (simultaneously update θ_0, θ_1)

  }

- New algorithm $(n \geq 1)$:

Repeat {

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

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

}

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

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

⋮

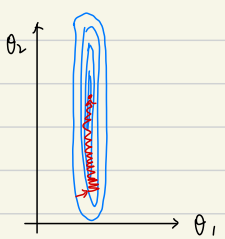
Gradient Descent in Practice I - Feature Scaling

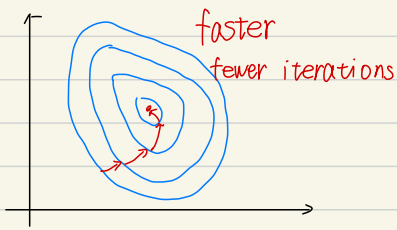
- Feature Scaling

Idea: Make sure features are on a similar scale.

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

x_2 = number of bedrooms (1-5)





$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$ $\quad x_2 = \frac{\text{number of bedrooms}}{5}$

$0 \le x_1 \le 1$ $\quad 0 \le x_2 \le 1$

Feature Scaling: Get every feature into approximately a $-1 \le x_i \le 1$ range

$x_0 = 1$

$0 \le x_1 \le 3$ ✓

$-2 \le x_2 \le 0.5$ ✓ $\quad -3$ to 3 ✓

$-100 \le x_3 \le 100$ ✗ $\quad -\frac{1}{3}$ to $\frac{1}{3}$ ✓

$-0.0001 \le x_4 \le 0.0001$ ✗

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

- Apply feature scaling and mean normalization:
  E.g. $x_1 = \frac{\text{size} - 1000}{2000}$ $\quad x_2 = \frac{\#\text{bedrooms} - 2}{5}$

  $-0.5 \le x_1 \le 0.5$ $\quad -0.5 \le x_1 \le 0.5$

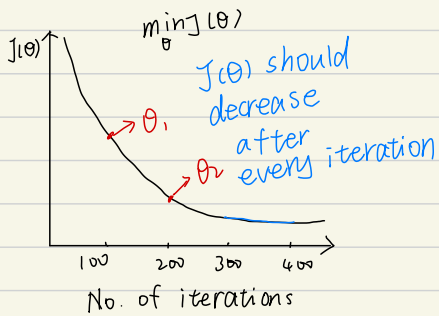
- $x_1 := \frac{x_1 - \mu_1}{s_1}$ ← avg value of x_1 in traing set
  ↖ range (max - min) (or standard deviation)

Gradient descent in Practice II - Learning Rate

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

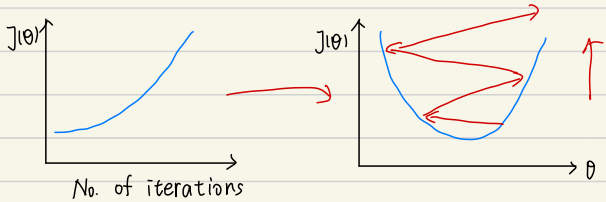
- "Debugging": How to make sure gradient descent is working correctly
- How to choose learning rate α.

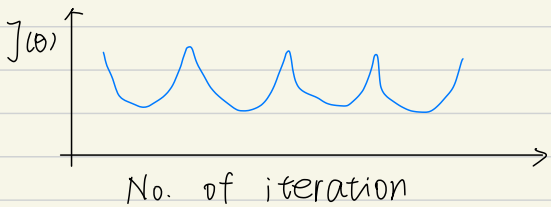
- Making sure gradient descent is working correctly.



- Example automatic convergence test: Declare convergence if $J(\theta)$ decreases by less than ε in one iteration. (ε might be 10^{-3}, but is hard to decide)

- Gradient descent not working. Use smaller α:





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

- Summary:
  - If α is too small: slow converge
  - If α is too large: $J(\theta)$ may not decrease on every iteration; may not decrease. (slow converge also possible)

- To choose α, try

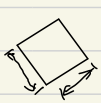
$\cdots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \cdots$

(×3, ≈×3, ×3, ≈×3)

Features and Polynomial Regression

- House prices prediction

$h_\theta(x) = \theta_0 + \theta_1 \times \underbrace{\text{frontage}}_{x_1} + \theta_2 \times \underbrace{\text{depth}}_{x_2}$



- We can combine multiple features into one. For example, we can combine x_1 and x_2 into a new feature x_3 by taking $x_1 \cdot x_2$

  Area x = frontage × depth

  $h_\theta(x) = \theta_0 + \theta_1 x$ ← land area

- Polynomial regression

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

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

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

$= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3$

$x_1 = (\text{size})$ size: 1 - 1000

$x_2 = (\text{size})^2$ size^2: 1 - 1000,000

$x_3 = (\text{size})^3$ size^3: 1 - 10^9

- Choice of features

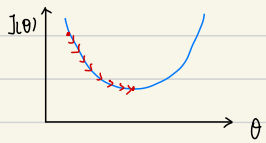
$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2$

$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{\text{size}}$

Computing Parameters Analytically

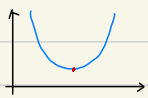
Normal Equation

- Gradient Descent



- Normal equation: Method to solve for θ analytically.

- Intuition: If 1D ($\theta \in \mathbb{R}$)



$J(\theta) = a\theta^2 + b\theta + c$

$\frac{d}{d\theta} J(\theta) = \cdots \overset{\text{set}}{=} 0$, solve for θ.

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

$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots \overset{\text{set}}{=} 0$ (for every j) $\quad$ solve for $\theta_0, \theta_1, \cdots, \theta_n$

- Example: $m = 4$

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

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

$m \times (n+1)$ $\qquad$ m-dimesional vector

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

- m examples $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$;
  n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \qquad X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

(design matrix)

$m \times (n+1)$

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$ $\qquad$ $X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix}$ $\qquad$ $y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

$m \times 2$

- $\theta = (X^T X)^{-1} X^T y$
  $(X^T X)^{-1}$ is inverse of matrix $X^T X$

- Octave : `pinv(X'*X)*X'*y`

- $0 \le x_1 \le 1$
  $0 \le x_2 \le 1000$ ✓
  $0 \le x_3 \le 10^{-5}$ ✓

~~Feature Scaling~~

- m training examples, n features

Gradient Descent	Normal Equation
• Need to choose α	• No need to choose α
• Need many iterations	• Don't need to iterate
• Works well even when n is large	• Need to compute $(X^TX)^{-1}$
$O(kn^2)$ $n = 10^6$	$(n \times n)^{-1}$ $O(n^3)$
	• Slow if n is very large
	$n > 10^4$? $n = 100, 1000, 10000$]

Normal Equation Noninvertibility

- Normal equation : $\theta = (X^TX)^{-1}X^Ty$
- What if X^TX is non-invertible? (sigular / degenerate)

- Octave : `pinv(X'*X)*X'*y` } pinv ✓ even when non-invertible
  inv

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

- Redundant features (linearly dependent)
  E.g. x_1 = size in feet^2 $\quad 1m = 3.28$ feet
  x_2 = size in m^2 $\quad x_1 = (3.28)^2 x_2$
  Delete a feature that is linearly dependent with another

- Too many features (e.g. $m \leq n$) $\quad m = 10,\ n = 100$
  $\theta \in \mathbb{R}^{101}$
  Delete some features, or use regularization.

Week 3 Logistic Regression

Classification and Representation

Classification

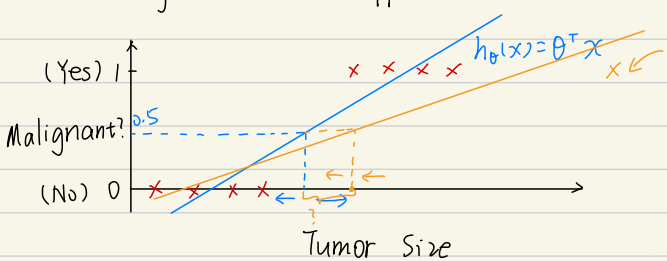
- Classification:
  - Email: Spam / Not Spam?
  - Online Transactions: Fraudulent (Yes / No)?
  - Tumor: Malignant / Benign?

$y \in \{0, 1\}$ $y \in \{0, 1, 2, 3\} \dots$

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

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

- Linear Regression (not applicable)



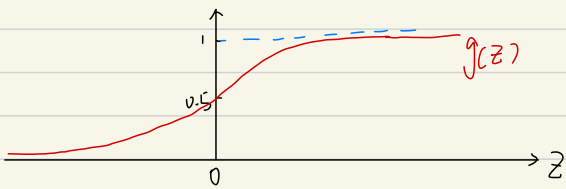
- Threshold classifier output $h_\theta(x)$ at 0.5:
  - If $h_\theta(x) \geq 0.5$, predict "$y = 1$"
  - If $h_\theta(x) < 0.5$, predict "$y = 0$"

- Classification: $y = 0$ or 1
  $h_\theta(x)$ can be > 1 or < 0
- Logistic Regression: $0 \leq h_\theta(x) \leq 1$ (classification)

Hypothesis Representation

- Logistic Regression Model: Want $0 \le h_\theta(x) \le 1$
- $h_\theta(x) = g(\theta^T x)$

  $g(z) = \frac{1}{1+e^{-z}}$ (Sigmoid function / Logistic function)
- $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$



- Interpretation of Hypothesis Output

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

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

  $h_\theta(x) = 0.7$: Tell patient that 70% chance of tumor being malignant.

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

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

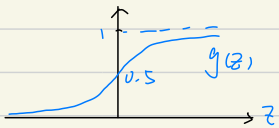
- $y = 0$ or 1: $P(y=0 \mid x; \theta) + P(y=1 \mid x; \theta) = 1$

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

Decision Boundary

- Logistic regression: $h_\theta(x) = g(\theta^T x) = P(y=1 \mid x;\theta)$

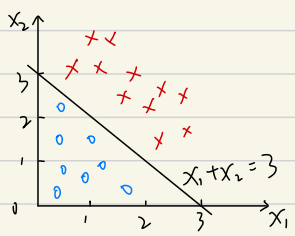
$$g(z) = \frac{1}{1+e^{-z}}$$



- Suppose predict "y=1" if $h_\theta(x) \geq 0.5$
  predict "y=0" if $h_\theta(x) < 0.5$
- $g(z) \geq 0.5$ when $z \geq 0$
  $h_\theta(x) = g(\theta^T x) \geq 0.5$ when $\theta^T x \geq 0$

- Decision Boundary: the line that seperates the area where y=0 and where y=1. Created by the hypothesis function.

- E.g. linear



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

$\theta_0 = -3 \quad \theta_1 = 1 \quad \theta_2 = 1 \quad \theta^T x$

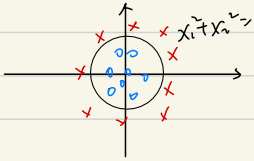
- Predict "y=1" if $-3 + x_1 + x_2 \geq 0$

$x_1 + x_2 \geq 3$

Boundary: $x_1 + x_2 = 3$

y=0: $x_1 + x_2 < 3$

- Non-linear decision boundaries



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

$\theta_0 = -1, \theta_1 = 0, \theta_2 = 0, \theta_3 = 1, \theta_4 = 1$

- Predict "y=1" if $-1 + x_1^2 + x_2^2 \geq 0$

$x_1^2 + x_2^2 = 1$

Logistic Regression Model

Cost Function

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

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $\quad x_0 = 1 \quad y \in \{0, 1\}$

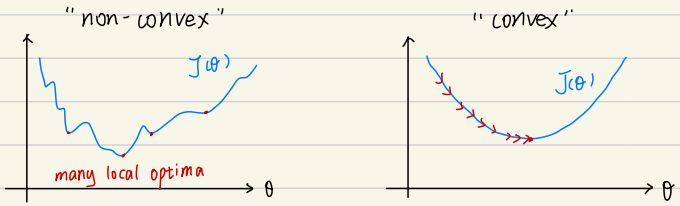
$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

- How to choose parameters θ?
- Cost Function

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

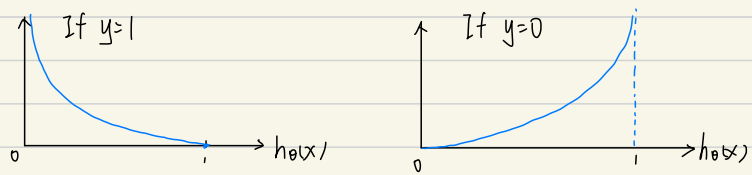
  Logistic regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x^{(i)}) - y^{(i)})$

- If choose $\text{cost}(h_\theta(x), y) = \frac{1}{2}(h_\theta(x) - y)^2$: non-convex



- Logistic regression cost function

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



- If $y=1$: $h_\theta(x)=1 \Rightarrow$ Cost $=0$
  $h_\theta(x) \to 0 \Rightarrow$ Cost $\to \infty$

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

Simplified Cost Function and Gradient Descent

Logistic regression cost function:

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

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

Note: $y=0$ or 1 always

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

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

  If $y=0$: $\text{Cost}(h_\theta(x), y) = -\log(1-h_\theta(x))$
- $J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$
- To fit parameters θ: $\min_\theta J(\theta)$
- To make a prediction given new x: Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

- Gradient Descent: Want $\min_\theta J(\theta)$:

  Repeat {

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

  } (simultaneously update all θ_j)

- Repeat { (Using calculus)

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

  $h_\theta = g(X\theta)$

  }
- Algorithm looks identical to linear regression
- Vectorized implementation: $\theta := \theta - \frac{\alpha}{m} X^T(g(X\theta) - \vec{y})$

Advanced Optimization

- Optimization Algorithm

  Cost function $J(\theta)$. Want $\min_\theta J(\theta)$.

  Given θ, we have code that compute: $J(\theta)$, $\frac{\partial}{\partial\theta_j}J(\theta)$ (for $j = 0, 1, \cdots, n$)
- Gradient Descent: Repeat{ $\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$ }

- Optimization algorithm: Given θ, we have code that can compute: $J(\theta)$ and $\frac{\partial}{\partial\theta_j}J(\theta)$ (for $j = 0, 1, \cdots, n$)

- Optimization algorithm: Gradient descent

Conjugate gradient
BFGS
L-BFGS

Advantages: No need to manually pick α

Often faster than gradient descent

Disadvantages: More complex

Example: $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$ $\quad \min_{\theta} J(\theta) : \theta_1 = \theta_2 = 5$

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

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

- Code:

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

users will provide a gradient — the maximum number of iteration

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

advanced — a pointer to function — $\theta \in \mathbb{R}^d, d \geq 2$

- theta = $\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ — theta(1), theta(2), theta(n+1)

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

Multiclass Classification

:

gradient(n+1) = [code to compute $\frac{\partial}{\partial \theta_n} J(\theta)$];

Multiclass Classification: One-vs-all

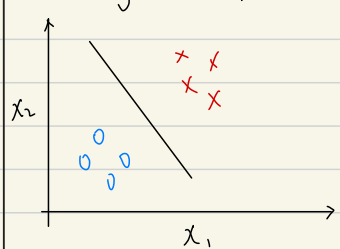
- Multiclass classification

Email foldering/tagging: Work, Friends, Family, Hobby
$y=1$ $y=2$ $y=3$ $y=4$

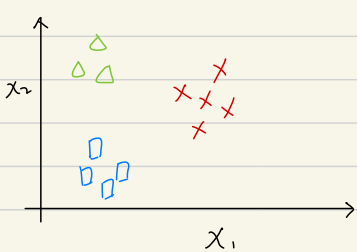
Medical diagrams: Not ill, Cold, Flu
$y=1$ 2 3

Weather: Sunny, Cloudy, Rain, Snow
$y=1$ 2 3 4

Binary classification

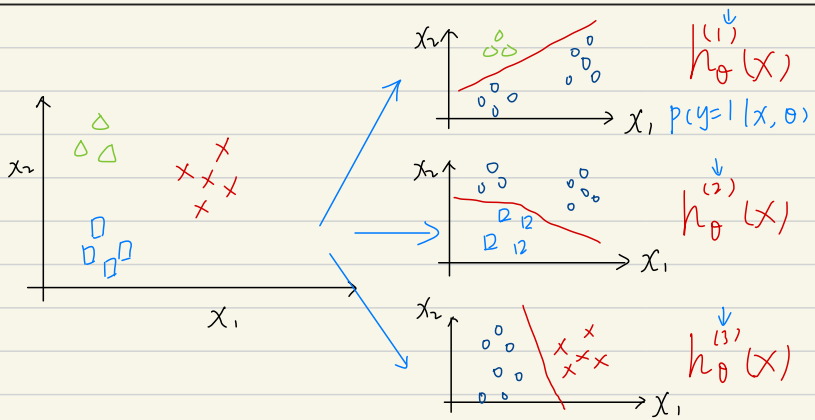


Multi-class classification



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

Class 1: △
Class 2: □
Class 3: ×



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

- One-vs-all

  Train a logistic regression classfier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y=i$.

  On a new input x, to make a prediction, pick the class i that maximizes $\max_i h_\theta^{(i)}(x)$

- $y \in \{0, 1, \cdots, n\}$

  $h_\theta^{(0)}(x) = P(y=0|x;\theta)$

  $\vdots$

  $h_\theta^{(n)}(x) = P(y=n|x;\theta)$

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

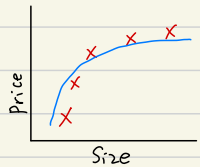
Solving the Problem of Overfitting

The Problem of Overfitting

- Example: Linear regression (housing prices)

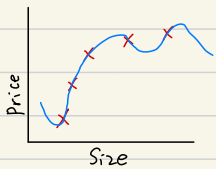
$\theta_0 + \theta_1 x$

"underfitting"
"high bias"



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

"just right"

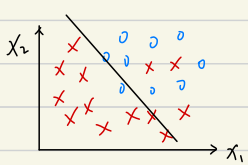


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

"overfitting"
"high varience"

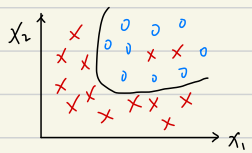
- Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \simeq 0$), but fail to generalize to new examples (predict prices on new examples)

- Example: Logistic regression g = sigmoid function



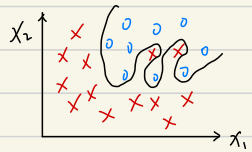
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

"underfitting"
"high bias"



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

"just right"

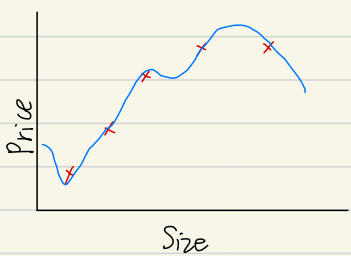


$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \cdots)$

"overfitting" "high variance"

- Addressing overfitting:

x_1 = size of house
x_2 = no. of bedrooms
x_3 = no. of floors
x_4 = age of house
⋮
x_{100}



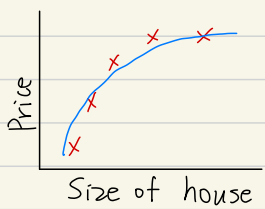
- Options:
  1. Reduce number of features.
     - Manually select which features to keep
     - Model selection algorithm (later in course)

Cost Function

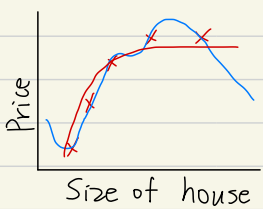
2. Regularization

- Keep all the features, but reduce magnitude/values of parameters θ_j.
- Works well when we have a lot of features, each of which contributes a bit to predicting y.

Intuition



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



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

- Suppose we penalize and make θ_3, θ_4 really small:

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

$\theta_3 \approx 0$ $\quad$ $\theta_4 \approx 0$

- Regularization: Small values for parameters $\theta_0, \theta_1, \cdots, \theta_n$
  - "Simpler" hypothesis
  - Less prone to overfitting

- Housing:

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

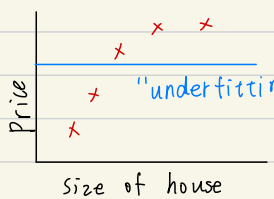
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

~~θ_0~~ $\theta_1, \theta_2, \cdots, \theta_n$

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

  regularization parameter

- In regularized linear regression, we choose θ to minimize $J(\theta)$. What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



$\theta_1 \approx 0 \quad \theta_2 \approx 0 \quad \theta_3 \approx 0 \quad \theta_4 \approx 0$

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

$\rightarrow h(x) = \theta_0$

Regularized Linear Regression

- Regularized linear regression

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

- Gradient descent

Repeat {

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

$$\theta_j := \theta_j - \alpha \underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x_0^{(i)} + \frac{\lambda}{m}\theta_j\right]}_{\frac{\partial}{\partial \theta_j} J(\theta)} \quad (j = \cancel{0}, 1, 2, \cdots, n)$$

regularized

}

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

$1 - \alpha\frac{\lambda}{m} < 1$: reducing θ_j on every update

- Normal equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \quad m \times (n+1) \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \mathbb{R}^m \qquad \rightarrow \min_\theta J(\theta)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) \overset{\text{set}}{=} 0 : \quad \theta = (X^TX + \lambda L)^{-1}X^Ty$$

$$L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \quad (n+1) \times (n+1)$$

- Non-invertibility (optional / advanced)

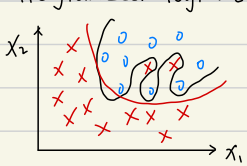
Suppose $m < n$, $\theta = (X^TX)^{-1}X^Ty$ — X^TX: non-invertible / singular

pinv, inv

If $\lambda > 0$, $\theta = (X^TX + \lambda L)^{-1}X^Ty$ — $X^TX + \lambda L$: invertible

Regularized Logistic Regression

- Reglarized logistic regression



$$h_\theta(x) = g(\theta_0 + \theta_1x_1 + \theta_2x_1^2 + \theta_3x_1^2x_2 + \theta_4x_1^2x_2^2 + \theta_5x_1^2x_2^3 + \cdots)$$

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

- Gradien descent

  Repeat {

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

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

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

  $(j = 0, 1, 2, \cdots, n)$ $\quad \frac{\partial}{\partial \theta_j} J(\theta)$

  }

- Advanced optimization

  fminunc(@costFunction, ···)

  $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ — θ_0 → theta(1), θ_1 → theta(2), θ_n → theta(n+1)

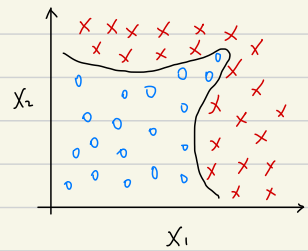
```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
        J(θ) = [-(1/m)Σ_{i=1}^{m} y^(i) log(h_θ(x^(i))) + (1-y^(i)) log(1-h_θ(x^(i)))] + (λ/2m)Σ_{j=1}^{n} θ_j^2
    gradient(1) = [code to compute ∂/∂θ_0 J(θ)];
        (1/m)Σ_{i=1}^{m}(h_θ(x^(i)) - y^(i)) x_0^(i)
    gradient(2) = [code to compute ∂/∂θ_1 J(θ)];
        (1/m)Σ_{i=1}^{m}(h_θ(x^(i)) - y^(i)) x_1^(i) + (λ/m)θ_1
    gradient(3) = [code to compute ∂/∂θ_2 J(θ)];
        (1/m)Σ_{i=1}^{m}(h_θ(x^(i)) - y^(i)) x_2^(i) + (λ/m)θ_2
    :
    gradient(n+1) = [code to compute ∂/∂θ_n J(θ)];
```

Week 4 Neural Networks: Representation

Motivations

Non-linear Hypothesis

Non-linear Classification



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

$n = 100$:

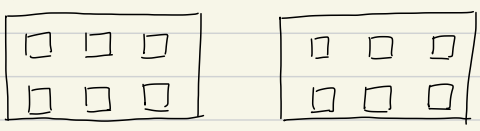
If: $x_1^2, x_1 x_2, x_1 x_3, \cdots x_1 x_{100}, x_2^2, x_2 x_3 \cdots$

≈ 5000 features $\approx \frac{1}{2} n^2$ $O(n^2)$

If: $x_1 x_2 x_3, x_1^2 x_2, \cdots, x_{10} x_{11} x_{17}, \cdots$

$\approx 170,000$ features $O(n^3)$

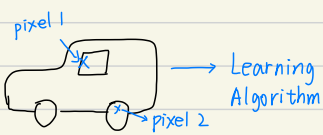
- Computer Vision: Car detection



Cars Not a car

Testing:

What is this?



50x50 pixel images → 2500 pixels

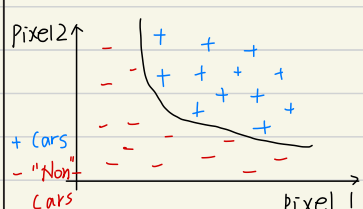
$n = 2500$ (7500 if RGB)

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

Quadratic features $(x_i \times x_j)$:

≈ 3 million features

expensive computation!



Neurons and the Brain

Neural Networks

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

The "one learning algorithm" hypothesis

- Auditory cortex learns to see (unable to listen)
- Somatosensory cortex learns to see (unable to touch)

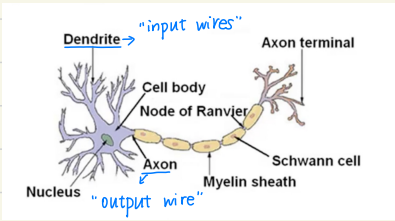
Sensor representations in the brain

- Seeing without your tongue
- Human echolocation (sonar)
- Haptic belt: Direction sense
- Implanting a 3rd eye

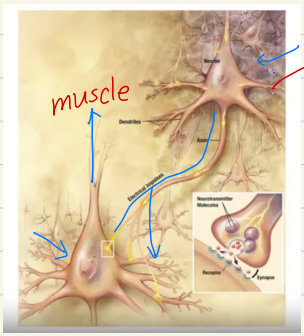
Neural Networks

Model Representation I

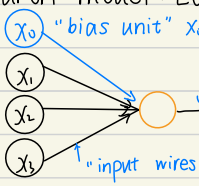
Neuron in the brain



Neurons in the brain



Neuron model: Logistic unit



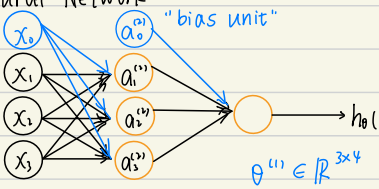
$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$ "weights" (parameters)

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

Model Representation II

Neural Network



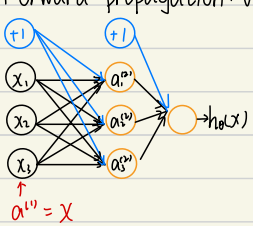
$a_i^{(j)}$ = "activation" of unit i in layer j

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

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

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

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \quad z_1^{(2)} \to a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \quad z_2^{(2)}$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \quad z_3^{(2)}$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) \quad z^{(3)}$$

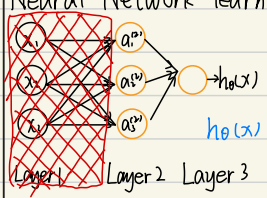
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$a_1^{(2)}$ $a_2^{(2)}$ $a_3^{(2)}$ — $\mathbb{R}^3$ — $\mathbb{R}^3$

$z^{(2)} = \Theta^{(1)} x = \Theta^{(1)} a^{(1)}$, $a^{(2)} = g(z^{(2)})$, Add $a_0^{(2)} = 1$ → $a^{(2)} \in \mathbb{R}^4$

$z^{(3)} = \Theta^{(2)} a^{(2)}$, $h_\Theta(x) = a^{(3)} = g(z^{(3)})$

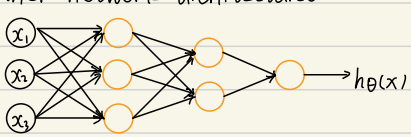
Neural Network learning its own features



$h_\Theta(x) = g(\Theta_{1,0}^{(2)} a_0^{(2)} + \Theta_{1,1}^{(2)} a_1^{(2)} + \Theta_{1,2}^{(2)} a_2^{(2)} + \Theta_{1,3}^{(2)} a_3^{(2)})$

logistic regression?

Other network architectures



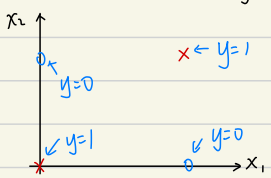
Layer 1 (Input) Layer 2, Layer 3 (Hidden layer) Layer 4 (Output)

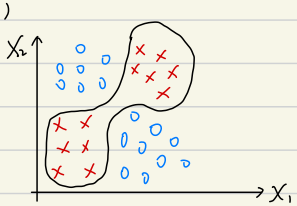
Applications

Examples and Intuitions I

Non-linear classification example : XOR / XNOR

x_1 and x_2 are binary (0 or 1)





$y = x_1$ XOR x_2

x_1 XNOR x_2

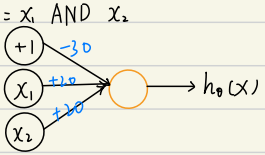
↓

Not (x_1 XOR x_2)

Simple example : AND

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

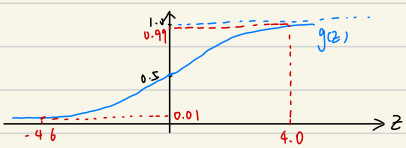
$y = x_1$ AND x_2



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

$\theta_{10}^{(1)}$ $\theta_{11}^{(1)}$ $\theta_{12}^{(1)}$

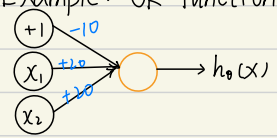
$h_\theta(x) \approx x_1$ AND x_2



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

Examples and Intuitions II

Example: OR function



$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$

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

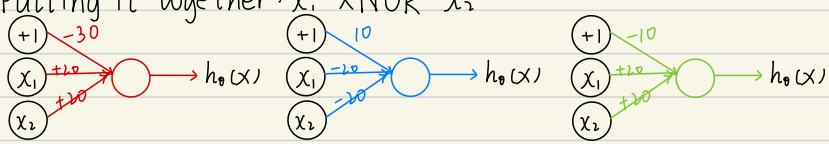
Negation: NOT x_1

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

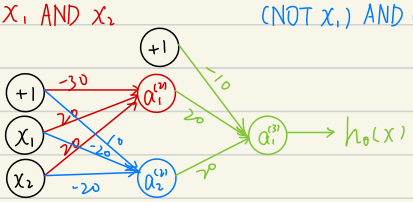
x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

(NOT x_1) AND (NOT x_2) $= 1$ iff $x_1 = x_2 = 0$

Putting it together: x_1 XNOR x_2



x_1 AND x_2 | (NOT x_1) AND (NOT x_2) | x_1 OR x_2



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

AND: $\Theta^{(1)} = [-30 \quad 20 \quad 20]$

NOR: $\Theta^{(1)} = [10 \quad -20 \quad -20]$

OR: $\Theta^{(1)} = [-10 \quad 20 \quad 20]$

> AND and NOR: $\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$

→ OR: $\Theta^{(1)} = [-10 \quad 20 \quad 20]$

$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow [a^{(3)}] \rightarrow h_\Theta(x)$

$a^{(2)} = g(\Theta^{(1)} \cdot x)$

$a^{(3)} = g(\Theta^{(2)} \cdot a^{(2)})$

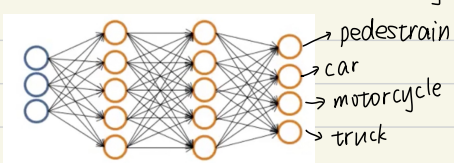
$h_\Theta(x) = a^{(3)}$

- Video: Handwritten digit classification

Multiclass Classification

Multiple output units: One-vs-all

Pedestrain Car Motorcycle Truck



$h_\Theta \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrain when car when motorcycle

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

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

pedestrain car motorcycle truck

previously: $y \in \{1, 2, 3, 4\}$

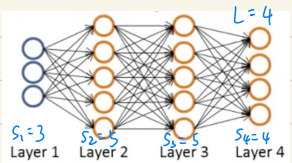
$(x^{(i)}, y^{(i)})$ $h_\Theta(x^{(i)}) \approx y^{(i)}$ ($\in \mathbb{R}^4$)

Week 5 Neural Networks: Learning

Cost Function and Backpropagation

Cost Function

Neural Network (Classification)



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

L = total no. of layers in network

s_l = no. of units (not counting bias unit) in layer l.

- Binary classification

  $y = 0$ or 1, 1 output unit, $h_\theta(x) \in \mathbb{R}$. $s_L = 1$, $k = 1$.

- Multi-class classification (k classes)

  $y \in \mathbb{R}^k$, k output units $h_\theta(x) \in \mathbb{R}^k$, $s_L = k$ $(k \geq 3)$

  E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

Cost function:

- Logistic regression:

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

- Neural Network:

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

$\theta^{(l)}$: $s_{l+1} \times s_l$ matrix

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 square of all the individual Θs in the entire network

Backpropagation Algorithm

Gradient computation

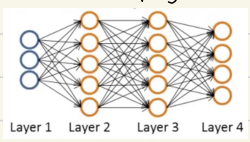
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\Theta(x^{(i)})_k + (1-y_k^{(i)}) \log (1-h_\Theta(x^{(i)})_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}} (\Theta_{ji}^{(l)})^2$$

$\min_\Theta J(\Theta)$

Need code to compute: $J(\Theta)$ & $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ $\quad \Theta_{ij}^{(l)} \in \mathbb{R}$

Given one training example (x, y):

Forward propagation: $a^{(1)} = x$



$a^{(1)}$ $a^{(2)}$ $a^{(3)}$ $a^{(4)}$

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

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

$z^{(3)} = \Theta^{(2)} a^{(2)}$

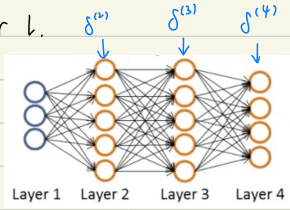
$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)

$z^{(4)} = \Theta^{(3)} a^{(3)}$

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

Backpropagation algorithm

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



For each output unit (layer $L=4$)

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

formula !

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

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

NO $\delta^{(1)}$

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

$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$ (ignory λ; if $\lambda = 0$)

Backpropagation algorithm

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

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

For $i = 1$ to m $\quad (x^{(i)}, y^{(i)})$

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

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

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

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

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

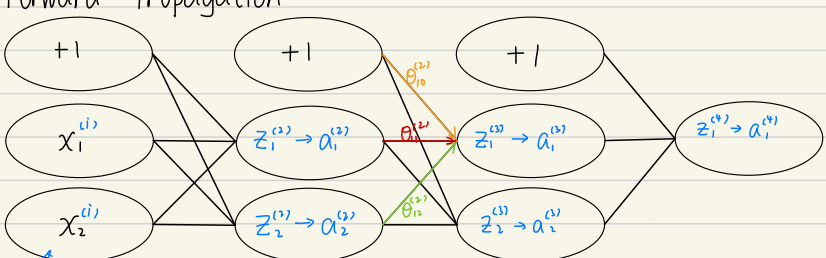
$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}$ if $j \neq 0$

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

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

Backpropagation Intuition

Forward Propagation



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

$z_1^{(2)} = \theta_{10}^{(1)} \times 1 + \theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)}$

What is backpropagation doing? → $k=1$

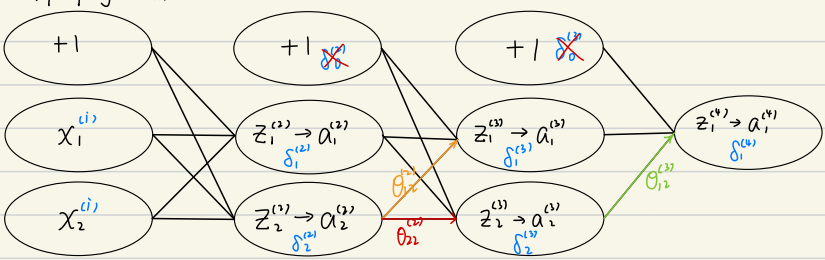
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\Theta(x^{(i)})\right) + (1-y^{(i)}) \log\left(1-h_\Theta(x^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

Focusing on a single example $x^{(i)}, y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$): $\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log\left(1-h_\Theta(x^{(i)})\right)$

(Think of $\text{cost}(i) \approx \left(h_\Theta(x^{(i)}) - y^{(i)}\right)^2$

→ how well is the network doing on example i?

Backpropagation



$\delta_j^{(l)}$ = "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 $j \geq 0$),

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

$\delta_1^{(4)} = a_1^{(4)} - y^{(i)}$ $\quad \delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}$ $\quad \delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)}$

Backpropagation in Practice

Implementation Note: Unrolling Parameters

Advanced optimization

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

gradient: $\mathbb{R}^{n+1}$; theta: $\mathbb{R}^{n+1}$ (vectors)

Neural Network ($L=4$):

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

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

"Unroll" into vectors

- Example

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

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

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

```
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);
```

(thetaVec: vector; Theta1(:): $\Theta^{(1)}$, all values)

- Learning Algorithm

Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$

Unroll to get initialTheta to pass to:

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

function [jval, gradientVec] = costFunction(thetaVec)

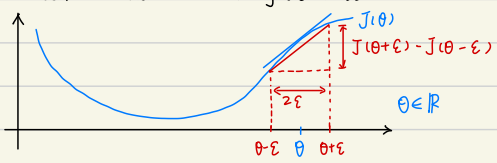
From thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ reshape

Use forward prop / back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\theta)$

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

Gradient Checking

Numerical estimation of gradients



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

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

Parameter vector θ

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

$\theta = [\theta_1, \theta_2, \theta_3, \cdots, \theta_n]$

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

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

$$\vdots$$

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

for i = 1:n,

thetaPlus = theta;

thetaPlus(i) = thetaPlus(i) + EPSILON;

thetaMinus = theta;

thetaMinus(i) = thetaMinus(i) - EPSILON;

gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2*EPSILON)

$\downarrow \frac{\partial}{\partial \theta_i} J(\theta)$

end;

$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \varepsilon \\ \vdots \\ \theta_n \end{bmatrix}$ $\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i - \varepsilon \\ \vdots \\ \theta_n \end{bmatrix}$

Check that gradApprox ≈ DVec ← From backpop

Radom Initialization

Initial value of θ

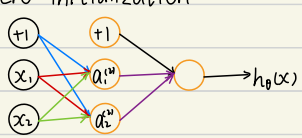
For gradient descent and advanced optimization method, need initial value for θ.

optTheta = fminunc(@costFunction, initialTheta, options)

Consider gradient descent:

Set initialTheta = zeros(n,1)?

Zero initialization



$\Theta_{ij}^{(1)} = 0$ for all i, j, l

$\rightarrow a_1^{(2)} = a_2^{(2)}$, $\delta_1^{(2)} = \delta_2^{(2)}$

$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$ $(\Theta_{01}^{(1)} = \Theta_{02}^{(1)})$

After each update, parameters corresponding to inputs going into each of two hidden units are identical.

Random initialization: Symmetry breaking

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

E.g. Theta1 = rand(10, 11) * (2*INIT_EPSILON) - INIT_EPSILON;

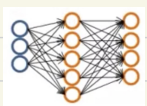
$[-\varepsilon, \varepsilon]$ random 10×11 matrix (between 0 and 1)

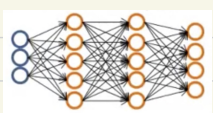
Theta2 = rand(1, 11) * (2*INIT_EPSILON) - INIT_EPSILON;

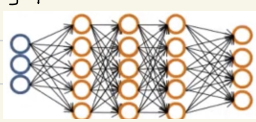
Putting it together

- Training a neural network

Pick a network architecture (connectivity pattern between neurons)







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

No. of output units: Number of classes

~~$y \in \{1, 2, 3, \cdots, 10\}$, $y = 5$~~ $y = \begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix}, \cdots, \begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix}$

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

- Training a neural network

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

for $i = 1:m$ {

Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \cdots, L$)

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

}

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.

Then disable gradient checking code

6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ.

$J(\Theta)$ - non-convex

Week 6 Advice for Applying Machine Learning

Evaluating a Learning Algorithm

Deciding what to try next

Debugging a learning algorithm:

Having implemented regularized linear regression to predict housing prices: $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{m}\theta_j^2\right]$

What if it makes unacceptably large errors in its prediction when we test the hypothesis on a new set of houses?

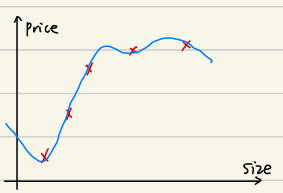
- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features (x_1^2, x_2^2, x_1x_2, etc.)
- Try decreasing / increasing λ

Machine learning diagnostic:

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

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

Evaluating a hypothesis



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

Fails to generalize to new examples not in training set.

Dataset:

	Size	Price
Training set (70%)	2104	400
	1600	330
	2400	369
	1416	232
	3000	540
	1985	300
	1534	315
Test set (30%)	1427	199
	1380	212
	1494	243

random order

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

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

m_{test} = no. of test examples

Training/testing procedure for linear regression

- Learn parameter θ from training data (minimizing training error $J(\theta)$)
- Compute test set error: $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$

Training/testing procedure for logistic regression

- Learn parameter θ from training data
- Compute test set error:

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

- Misclassification error (0/1 misclassification error):

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

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

Model Selection and Train / Validation / Test Sets

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

Model selection

d = degree of polynomial

$d=1$ 1. $h_\theta(x) = \theta_0 + \theta_1 x \rightarrow \theta^{(1)} \rightarrow J_{test}(\theta^{(1)})$

$d=2$ 2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)} \rightarrow J_{test}(\theta^{(2)})$

$\vdots$ $\vdots$ $\vdots$

$d=10$ 3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10} \rightarrow \theta^{(10)} \rightarrow J_{test}(\theta^{(10)})$

If choose $\theta_0 + \cdots + \theta_5 x^5$, how well does the model generalize? Report test set error $J_{test}(\theta^{(5)})$?

Problem: $J_{test}(\theta^{(5)})$ is likely to be an optimistic estimate of generalization error.

I.e. our extra parameter (d = degree of polynomial) is fit to test set.

Dataset:

	Size	Price	random order	
Training set 60%	2104	400		$(x^{(1)}, y^{(1)})$
	1600	330		⋮
	2400	369		$(x^{(m)}, y^{(m)})$
	1416	232		$(x_{cv}^{(1)}, y_{cv}^{(1)})$
	3000	540		⋮
	1985	300		$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$ → m_{cv} = no. of cv examples
cross validation set (CV) 20%	1534	315		$(x_{test}^{(1)}, y_{test}^{(1)})$
	1427	199		$(x_{test}^{(2)}, y_{test}^{(2)})$
test set 20%	1380	212		⋮
	1494	243		$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Train/validation/test error

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

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

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

Model selection

$d=1$ 1. $h_\theta(x) = \theta_0 + \theta_1 x \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$

⋮

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

If $\min = J_{cv}(\theta^{(4)})$, then choose $d = 4$.

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

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

Bias vs. Variance

Diagnosing Bias vs. Variance