WEEK 1

Machine learning definition:

Field of study that gives Computers the ability to learn without being explicitly programmed.

—Arthur Samuel

A computer is said to learn from experience E with respect to some class of tasks T and performance P, if its performance at tasks in T, as measured by P, improves with experience E.

—Tom Mitchell

Intro to supervised learning

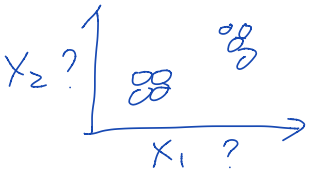
"right answers" given

Categorized as:

- Regression: Predict continuous valued output
- Classification: Discrete valued output

Intro to unsupervised learning

Approach problems with little or no ideas about what the results should look like

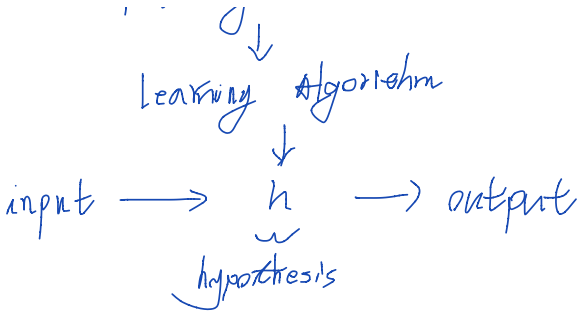


Supervised learning model

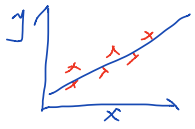
Training set

linear regression with one variable:

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



Cost function



θ_0, θ_1 to minimize: $\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

m: # training examples

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

y: actual value

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

(Squared error function)

Commonly used for regression problems

Intuition:

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

Linear regression with one variable —— Gradient descent

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

Outline:

start with some θ_0, θ_1

keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until hopefully end up with a minimum

repeat until convergence: {

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

}

α: learning rate

Simultaneously update θ_1, θ_0

Gradient decent for linear regression

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

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

$$\begin{cases} \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \\ \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \end{cases}$$

update θ_0, θ_1 simultaneously

Cost function for linear regression will always be a convex function (Bowl-shaped), 1 global/local min

"Batch" Gradient Descent

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

Linear ~~Algebra~~ Review

skip. I know ~~everything~~

WEEK 2

Multivariate Linear Regression

Notation: n : # of features

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

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

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

define $x_0 = 1$, then $x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$

also: $\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

then: $h_\theta(x) = \theta^T x$

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

Gradient descent:

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

Could be due to learning rate α too large, try use small α

Mathematicly, for sufficiently small α, $J(\theta)$ should decrease on every iteration

but if α is too small, gradient decent can be slow to converge

Features and Polynomial Regression

e.g. $h(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 = \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$

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

One should take care of feature scaling, since $x_1 \sim x_3$ takes every different ranges

Normal Equation:

Normal equation: method solve for θ analytically

$\frac{\partial J(\theta)}{\partial \theta_j} = 0 \Rightarrow \theta_0 \cdots \theta_n$

x_0	feature 1 x_1	feature 2 x_2	feature 3 x_3	feature 4 x_4	results y
1					
1					
1					
⋮					
1					

$$X = \begin{bmatrix} 1 & & & & \\ 1 & & & & \\ 1 & & & & \\ 1 & & & & \end{bmatrix}_{m \times (n+1)} \qquad y = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}_{m \times 1}$$

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

Generally:

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

n features

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

design matrix

It Does NOT matter whether using feature scaling or not in this case

Compare:

Gradient decent

- need to chose α
- needs many iterations
- works well even feature n is longer
- more general

Normal equation

- no need chose α
- No need of iteration
- need to compute $[X^TX]^{-1}$ (slow when n is large)
- linear regression only
- $[X^TX]^{-1}$ may non-invertible

what does $[X^TX]$ is non-invertible mean?

- Redundant features
- Too many features ($m \leq n$)

Vectorization:

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T x$$

WEEK 3

Logistic Regression

Regularization

WEEK 3

Classification

$y = 0$ or 1

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

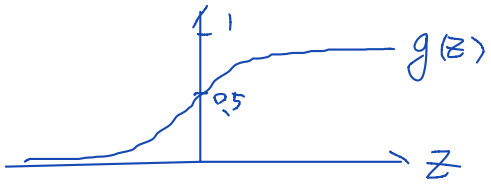
Logistic regression — hypothesis representation

model: want $0 \le h_\theta(x) \le 1$

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

$= \frac{1}{1 + e^{-z}}$ ← Sigmoid or logistic function

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



for binary classification: $h_\theta(x) = P(y=1 \mid x;\theta)$

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

Logistic regression — decision boundary

$h_\theta(x) = P(y=1 \mid \theta; x)$ { predict $y=1$ if $h_\theta(x) \geq 0.5$
- - - - $y=0$ if $h_\theta(x) < 0.5$

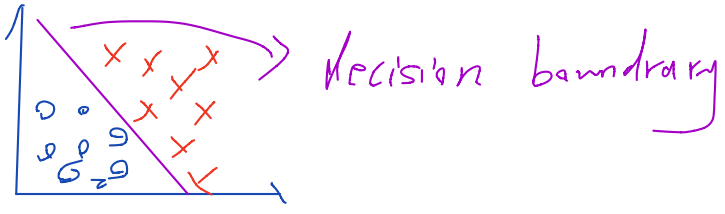
$g(z) = \frac{1}{1+e^{-z}}$

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

↳ $\theta^T x \geq 0$

$g(z) < 0.5$ when $z < 0$

↳ $\theta^T x < 0$



Logistic regression — cost function

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

$x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $x_0 = 1$, $y \in \{0, 1\}$

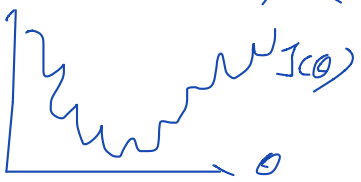
m train sets, n features

$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ how to choose parameters θ ?

Cost function

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

& if we use the above same cost function for Logistic regression,
the cost function will be "non-convex" (has many local minimums)

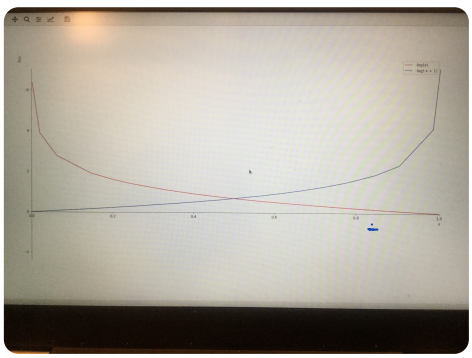


which is not garenteed to converge to global minimum
Thus we have to modify the cost function for logistic regression

defination:

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

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



Plot of cost

Simplified Cost function and gradient decerent

Logistic regression cost function: $J_\theta = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$

Simplification:

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

then:

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

To fit parameters θ: $\min_\theta J(\theta)$

To make a prediction giving new x: $h(\theta) = \frac{1}{1+e^{-\theta^T x}}$

Gradient descent

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

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

then:

$$\theta_j := \theta - \alpha \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad [\text{Sim update all } \theta_j]$$

Andrew Ng did put $\frac{1}{m}$ here, which I think is a mistake

Looks exactly the same as linear regression!

Except the definition of hypothesis $h_\theta(x)$ is diff.

Feature scaling also works for logistic regression!

Vectorized implementation:

$$\theta := \theta - \frac{\alpha}{m} X^T \cdot (g(X\theta) - \vec{y})$$

Logistic regression — adv. optimization

given θ, we have code that Compute:

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

Optimization algorithms

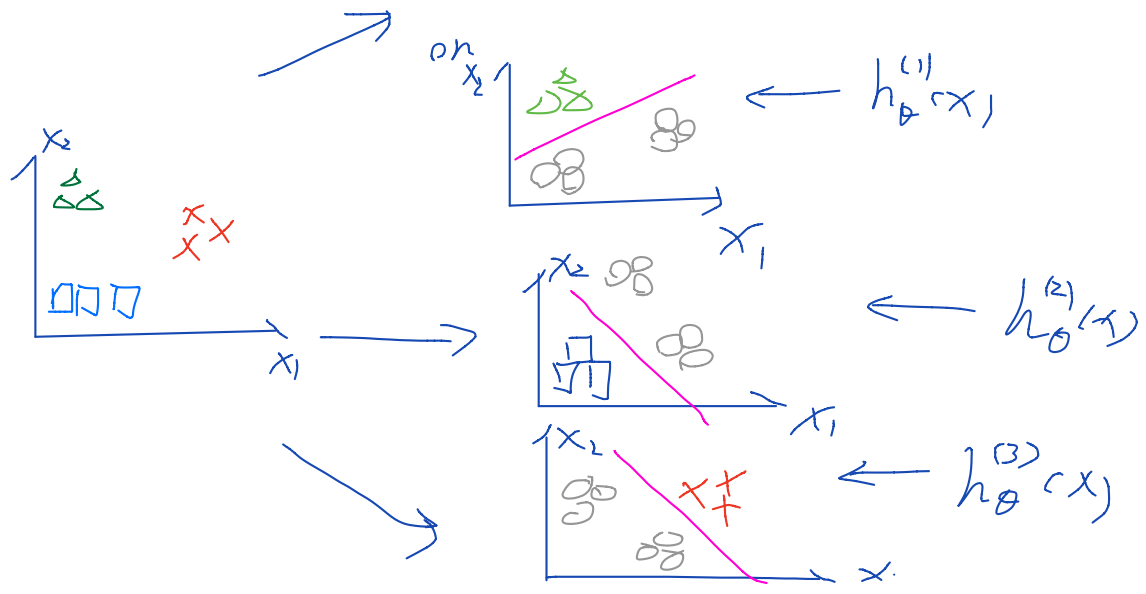
- Gradient decent
- Conjugate gradient
- BFGS
- L-BFGS

advantages:
- No need to manually pick α
- often faster than gradient decent

Disadvantages:
- more complex

Multi-class classification

one-vs-all (rest) classification



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

one-vs-all

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

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

Problem of over fitting

Overfitting:

If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) \approx 0$), but fail to generalize to new example

Addressing overfitting

Options:

1. Reduce number of features
   a. Manually select which features to keep
   b. Model selection algorithm (later)
2. Regularization
   a. keep all features, but reduce magnitude / values of θ_j
      ↳ Works well when having a lot of features, each of which contributes a bit to prediction y

Regularization — cost function

Regularization:

Small values for parameters $\theta_0, \cdots \theta_n$

↳ "simpler" hypothesis's

↳ less prone ~~to~~ over fitting

regularized Cost function

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

λ: regularization parameter

omitted θ_0 convention add θ_0 doesn't make any diff

λ: too larger ⟶ underfit .

Regularized linear regression:

① gradient decent:

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

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

$(j = ~~0~~, 1, 2, 3, \cdots n)$

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

usually α, λ small, m large, so $1 - \alpha\frac{\lambda}{m} < 1$

② Normal Equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}_{m \times n+1} \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$\min_\theta J(\theta)$

↳ $\theta = (X^TX + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \end{bmatrix})^{-1}X^Ty$

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

Non-invertibility

if $m \le n$, $[X^T X]$ non-invertible

if $\lambda > 1$, then $(X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & 1 \end{bmatrix})^{-1}$ is garanted to be invertiable

Regularized logistic regression

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

regularize

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

Gradient descent:

Same as linear regression, except that hypothesis's func $h_\theta(x)$ is different.

WEEK 4

- Neural Networks: Representation

WEEK 4 Neural Networks

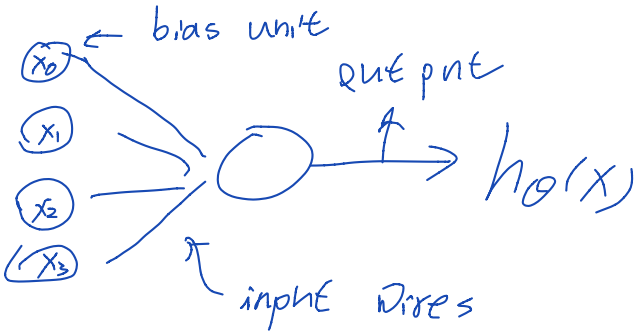
Neural networks motivation

- Complex regression
- Large number of features

Origins: Algorithms ~~that~~ try to mimic ~~the~~ brain

Somatosensory Cortex "one learning hypothesis"

Neural Networks representation



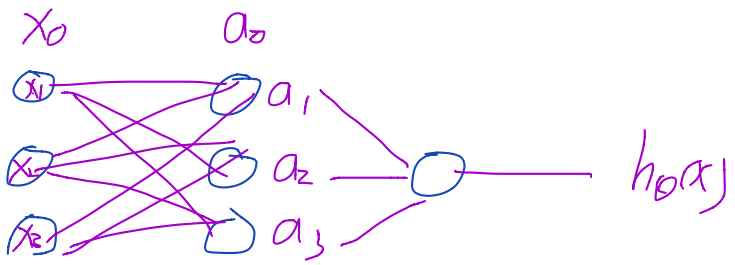
Neuron model: Logistic unit

terminology:

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

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

Neural Network:



layer 1 → Input Layer; layer 2 → Hidden Layer; layer 3 → Output Layer

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

$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) = z_1^{(2)}$

$a_2^{(2)} = g(\Theta_{20}^{(1)} \cdots \Theta_{21}^{(1)} + \Theta_{22}^{(1)} \cdots \Theta_{23}^{(1)} \cdots) = z_2^{(2)}$

$a_3^{(2)} = \cdots = z_3^{(2)}$

$\Theta^{(1)} \in \mathbb{R}^{3 \times 4}$

* if network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

Forward Propagation: Vectorized Implementation

vectorize:

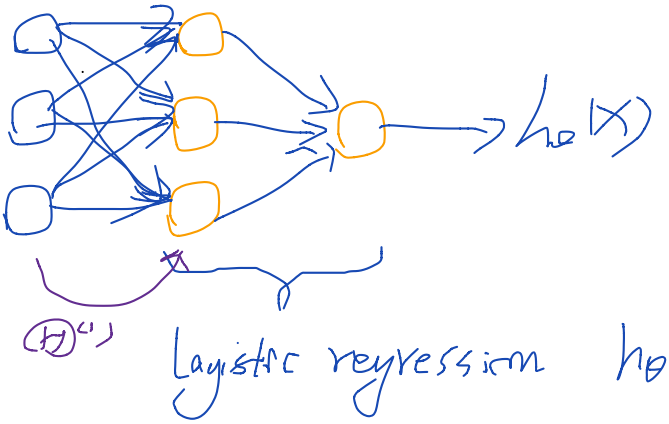
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \quad a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

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

also define $a^{(1)} = x$, add $a_0^{(2)} = 1$

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

Neural Network learning its own features



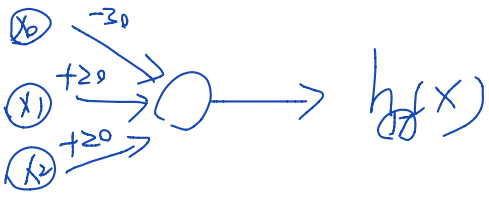
Logistic regression $h_\theta(x) = g\left(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3\right)$

$= g\left(\theta_0 a_0^{(2)} + \theta_1 a_1^{(2)} + \theta_2 a_2^{(2)} + \theta_3 a_3^{(2)}\right)$

Neural network learning its own features

Examples

(1) Logic AND function

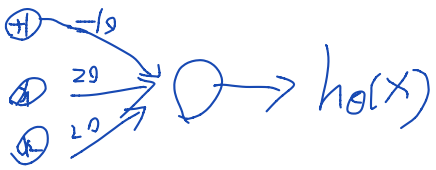


$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

g : sigmoid function

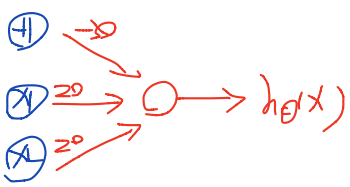
x_1	x_2	$h_\theta(x)$
0	0	0
0	1	0
1	0	0
1	1	1

(2) Logic OR function

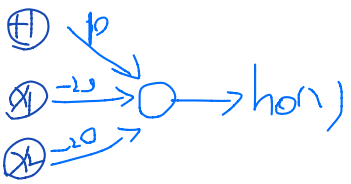


(3) Logic NOT

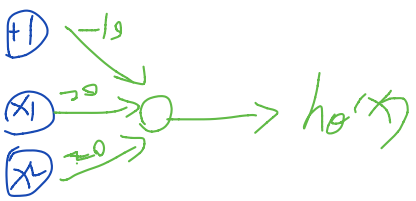
(4) XNOR



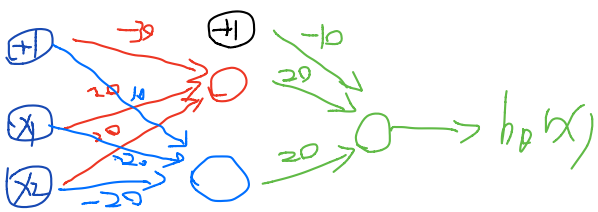
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2



XNOR !!!

Multi-class classification

output = $\begin{bmatrix}1\\0\\0\\\vdots\end{bmatrix}, \begin{bmatrix}0\\1\\0\\\vdots\end{bmatrix}, \cdots\cdots$

Neural Network classification problem

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

L: total no. of layers in the network

s_l: no. of units (exclude bias unit) in layer l

binary classification: $y = 0$ or 1 — 1 output unit

Multi-class classification (K classes) $\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix} \cdots \begin{bmatrix}0\\0\\\vdots\\0\\1\end{bmatrix}$ K output units

logistic regression

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

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

Neural Networks: Backpropagation algorithm

Gradient Computation: need $J(\Theta)$ $\quad \frac{\partial J(\Theta)}{\partial \Theta_{ji}^{(l)}}$

e.g. 1 training example, 4 layers, forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)} \qquad a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)} \qquad a^{(4)} = h_\Theta(x) = g(z^{(4)})$$

Backpropagation algorithm

intuition: $\delta_j^{(l)}$ = "error" of node j on layer l

e.g. for layer 4: $\delta_j^{(4)} = a_j^{(4)} - y_j$

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

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

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

Mathematically: $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$ (ignoring 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) (use to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

for $i = 1$ to m:

↳ set $a^{(1)} = x^{(i)}$ (input layer)

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

↳ using $y^{(i)}$, Compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

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

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

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

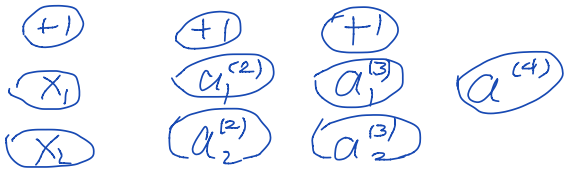
$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

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

→ use for gradient decent or other algorithms

Backpropagation intuition

forward propagation



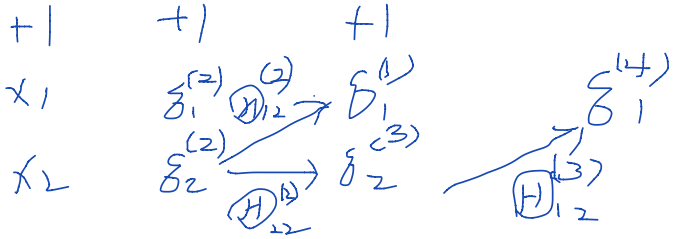
$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)}$$

backpropagation

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

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

formally: $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$



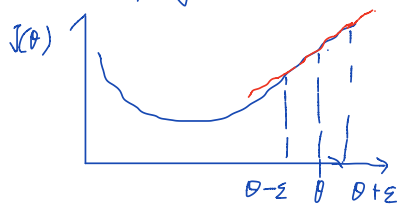
$$\delta^{(4)} = y^{(i)} - a_1^{(4)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta^{(4)}$$

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

Gradient Checking

Numerical estimation of gradients



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

$$\varepsilon \sim 10^{-4}$$

Parameter vector θ

$\theta \in \mathbb{R}^n$ (unrolled version of $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$)

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

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

$$\frac{\partial}{\partial\theta_2}J(\theta) \approx \cdots$$

$$\vdots$$

$$\frac{\partial}{\partial\theta_n}J(\theta) \approx \cdots$$

check that above approx $\approx \frac{\partial}{\partial\theta_j^{(l)}}J(\theta)$ from back propagation

Implementation Note

1) Implement backprop. to compute DVec (unrolled $D^{(1)}, D^{(2)}, \cdots$)
2) Implement numerical gradient check to compute gradApprox
3) Make sure they give similar values
4) Turn off gradient checking, using backprop code for learning.

Random Initialization

Initial value of θ

☆. Initialize all parameters to 0 when training a

Neural network is NOT OK !

(after each update, parameters corresponding to inputs going into each of two hidden units are identical

This prevents your neural network from learning anything interesting)

Random Initialization: Symmetry breaking

initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

Put it together

1) Pick a network architecture

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

No. of output units: number of classes

No. of hidden layers:

reasonable default: 1 hidden layer

or if >1 hidden layer, have same number of hidden units in each layer (usually the more the better)

2) Randomly initialize weights Θ

3) Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$

4) Implement code to compute cost function $J(\Theta)$

5) Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

for $i = 1 : m$ $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$

perform forward prop and backprop using example $(x^{(i)}, y^{(i)})$. get activations $a^{(l)}$ and delta $\delta^{(l)}$

for $l = 2, \cdots, L$

6) Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ from backprop and numerical estimation, then disable the gradient checking

7) Use gradient decent or adv. method with backprop to try to minimize $J(\Theta)$ as a function of parameters Θ

Decide What to Try Next

what to do if getting large errors in prediction ?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features ($x_1^2, x_2^2, x_1^3, x_2^3, \cdots$)
- Try decreasing/increasing λ

Machine Learning diagnostic

diagnostic: A test 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

Evaluate a hypothesis

Split training data into training set and test set to avoid overfitting

Training/testing procedure for linear regression

- Learn parameter θ from training set (70% of training data, e.g.)
- Compute test set error

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

(logistic regression is similar)

- Misclassification error:

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

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

Model selection and training / validation / test sets

Model selection:

choose right degree of polynomial:

1. $h_\theta(x) = \theta_0 + \theta_1 x \rightarrow (H)^{(1)}$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow (H)^{(2)}$
⋮ ⋮ ⋮
10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10} \rightarrow (H)^{(10)}$

$d :=$ degree of polynomial

choose d to minimize the error $J(\theta)$ in test set is NOT right, Since d might just be optimized for this specific test set.,

usually choose:

training data
- 60% Traing set $(x^{(i)}, y^{(i)})$
- 20% cross validation set (CV) $(x_{cv}^{(i)}, y_{cv}^{(i)})$
- 20% test set $(x_{test}^{(i)}, y_{test}^{(i)})$

Errors: $J_{train}(\theta)$, $J_{cv}(\theta)$, $J_{test}(\theta)$

choose the degree of polynomial d to get miniminm $J_{cv}(\theta)$ then use the θ from same d to calculate $J_{test}(\theta)$ which gives right test set error

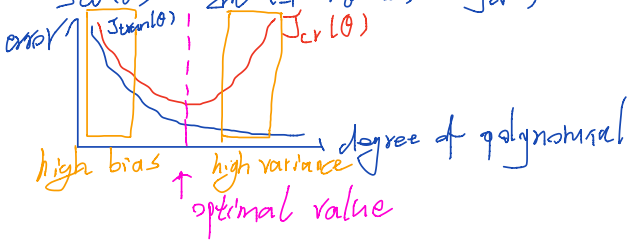
Diagnosing bias VS. variance

High bias: under fit

High variance: over fit

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

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



High Bias (underfit): $J_{train}(\theta)$ will be high, $J_{cv}(\theta) \approx J_{train}(\theta)$

High variance (over fit): $J_{train}(\theta)$ will be low, $J_{cv}(\theta) >> J_{train}(\theta)$

Regularization and bias/variance

Linear regression with regularization

Model: $\begin{cases} h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x^3 + \theta_4 x^4 \\ J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2 \end{cases}$

Large λ: $\lambda = 10000$, $\theta_1 \approx 0$, $\theta_2 \approx 0$, $\cdots \theta_4 \approx 0$ $h_\theta(x) = \theta_0 \Rightarrow$ high bias

Small λ: $\lambda \approx 0 \Rightarrow$ high variance (over fit)

intermediat $\lambda \Rightarrow$ Just right

Choosing the regularization parameter λ

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

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

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

$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x^{(i)}) - y^{(i)})^2$ — no regularization

$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$ ← term involved

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

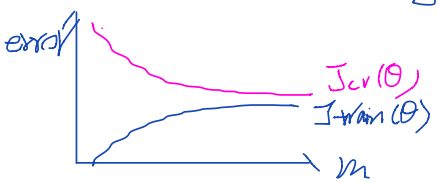
choose λ corresponding to smallest $J_{cv}(\theta)$, this is the proper value of λ

To see the error of this hypothesis, calculate $J_{test}(\theta)$

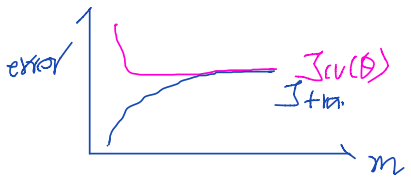
Bias/Variance as a function of the regularization Parameter

Learning Curves

Plot $J_{train}(\theta)$ and $J_{cv}(\theta)$ V.S. training set size m



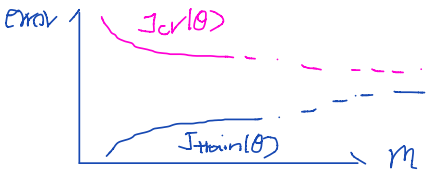
High bias



J_{cv} saturates pretty soon, $J_{train} \approx J_{cv}(\theta)$ by ~~the~~ end

* if a learning algorithm is suffering from high bias, getting more training data will not help much

High variance



* if a learning algorithm is suffering from high variance (overfitting), getting more training data is likely to help

Deciding what ~~to~~ to try next - revisit

Debugging a learning algorithm

Suppose you implemented regularized linear regression, however when you apply ~~this~~ hypothesis on a new ~~test~~ sets, you find it makes unacceptable large errors. What should you try next?

- Get more training examples
- Try smaller sets of features
- Try increasing λ

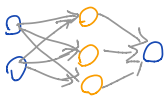
} fix high variance (overfitting) problems

- Try getting additional features

- Try adding polynomial features
- Try decreasing λ

} fix high bias (underfitting) problems

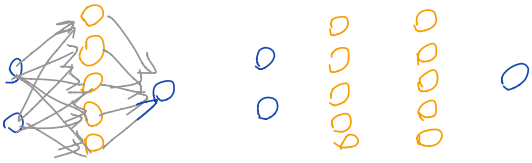
Neural Networks and overfitting



Small neural network:

- fewer parameters
- more prone to underfitting
- Computationally cheaper

Large neural network



- more parameters
- more prone to overfitting
- Computationally more expensive

Use regularization (λ) to address overfitting

PREFERED !!

Machine Learning System Design

Prioritizing what to work on: Spam classification example

Error Analysis

Recommended Approach

- Start with a simple algorithm that you can implement quick and dirty, and test it on your cross-validation data
- Plot learning curves to decide if more data, more features, etc. are likely to help
- Error analysis: Manually exam the examples (in CV data set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

Error metrics for skewed classes

Cancer classification example

Train logistic regression model $h_\theta(x)$, find that you got 1% error on test set (99% correct diagnostics)

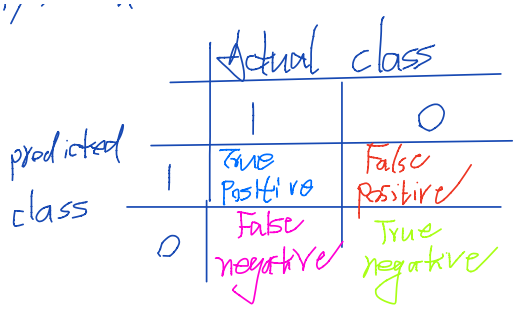
BUT if in the training set only 0.5% of people have cancer, prediction $y = 0$ (always) will give only 0.5% error, which seems better than "learned" predictions

⇓

Skewed classes

Problem: 99.5% accuracy is better than 99%??

Precision / Recall



$y=1$ in presence of rare class that we want to detect.

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

$$\frac{\text{True Positives}}{\#\text{ of predicted positive}} = \frac{\text{True positive}}{\text{Tru pos + False pos}}$$

Recall: (of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$\frac{\text{True Pos}}{\#\text{ actual pos}} = \frac{\text{True Pos}}{\text{True pos + false neg}}$$

Trading off precision and recall

Normally:

Logistical regression: $0 \le h_\theta(x) \le 1$

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

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

Suppose we want to predict $y=1$ only if very confident

$\rightarrow$ higher precision, lower recall: $0.5 \rightarrow 0.7$

Suppose we want to avoid missing too many cases of cancer

→ higher recall, lower precision 0.5 → 0.3

Vary value of threshold ⇒ trade off precision Vs recall

F1 Score (on CV-set)

How to compare precision (P) / recall (R) numbers

Average: $\frac{P+R}{2}$ → not good

F1 Score: $2\frac{PR}{P+R}$ P=0 or R=0 ⇒ F1=0
P=1 & R=1 ⇒ F1=1

Data for machine Learning

useful test: given the input x, can a human expert confidently predict y? (is features good enough?)

Large data rationale

algorithm with many features → low bias algorithms
..... Large training set → unlikely to overfit
→ $J_{train}(\theta)$ small
→ $J_{train}(\theta) \approx J_{test}(\theta)$

Combine above two.

$J_{test}(\theta)$ will be small(

$\Rightarrow$ have many features + Large training set

Large Margin Classification

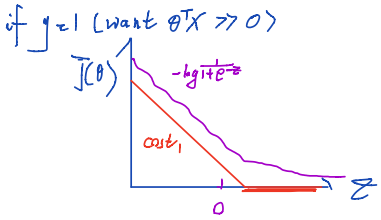
Optimization Objective

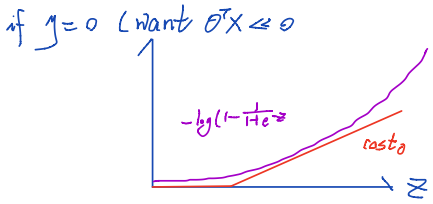
Alternative view of logistic regression

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

$$J(\theta) = -(y \log h_\theta(x) + (1-y)(\log(1-h_\theta(x)))$$

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





Support Vector machine (SVM)

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

$-\log h_\theta(x^{(i)}) \rightarrow \text{cost}_1(\theta^T x^{(i)})$, $-\log(1-h_\theta(x^{(i)})) \rightarrow \text{cost}_0(\theta^T x^{(i)})$

Support vector machine $\min_\theta \cancel{\frac{1}{m}} \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)})\text{cost}_0(\theta^T x^{(i)}) + \cancel{\frac{\lambda}{2m}}\sum_{j=1}^{n}\theta_j^2$

In logistic regression: $A + \lambda B$

In Support Vector machine $CA + B$ $(C = \frac{1}{\lambda})$

Cost function for SVM:

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

hypothesis of SVM

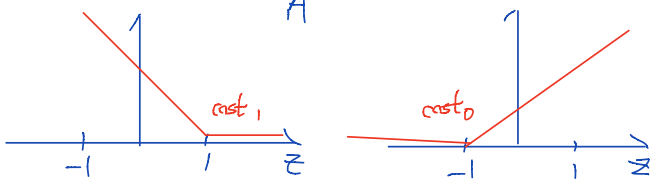
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ \end{cases}$$

... 1 0 else

Large Margin intuition

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

A



if $y=1$, we want $\theta^T x \geq 1$ (not just >0)

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

Suppose C very large, $\min_\theta CA + B \Rightarrow A \to 0$

whenever $y^{(i)} = 1$:

$\theta^T x \geq 1$ $\quad \min C \times 0 + \frac{1}{2}\sum_{i=1}^{n} \theta_j^2$

whenever $y^{(i)} = 0$

$\theta^T x \leq -1$, $\quad \min C \times 0 + \frac{1}{2}\sum_{i=1}^{n} \theta_j^2$

SVM Decision Boundary: linearly seperable case

Large Margin Classifier

C very large

SVM is a Large margin classifier

Mathematics behind Large Margin Classification

SVM decision boundary when C is very large:

(for simplicity, assume $n = 2$, $\theta_0 = 0$)

$$\min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}(\theta_1^2+\theta_2^2) = \frac{1}{2}\|\theta\| \qquad \theta=\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$$

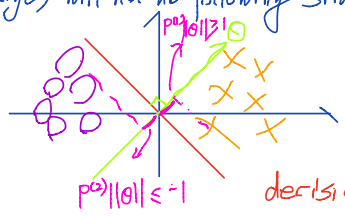
if $y^{(i)}=1$, $\theta^T x^{(i)} \geq 1 \Rightarrow p^{(i)}\cdot\|\theta\| \geq 1$

if $y^{(i)}=0$, $\theta^T x^{(i)} \leq -1 \Rightarrow p^{(i)}\cdot\|\theta\| \leq -1$

where $p^{(i)}$ is $x^{(i)}$ projection on to θ

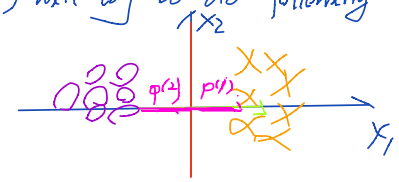
Now prove that SVM with large C with end up with large decision margin (large margin classification)

1) SVM (C large) will not do following small margin:



projection $p^{(i)}$ is small, means $\|\theta\|$ has to be large to make $p^{(1)}\|\theta\|\geq 1$ and $p^{(2)}\|\theta\|\leq -1$, but cost function want $\frac{1}{2}\|\theta\|$ to be small, so SVM (large C) will not end up with this situation

2) SVM (large C) will try to do following



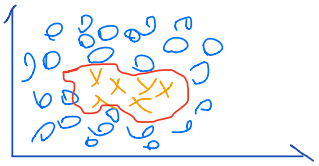
In this case $p^{(1)}$ is larger, to fulfil $p^{(1)}\|\theta\|\geq 1$ and $p^{(2)}\|\theta\|\leq -1$, $\|\theta\|$ doesn't need to be too large, which agrees the minimization of cost function $\min \frac{1}{2}\|\theta\|$

thus SVM (with large C) is a Large Margin Classification

Assumption $\theta_0 = 0$ make θ go through origin of above plot

Kernels

Nonlinear Decision Boundrary



predict $y=1$ if $\theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_1 X_2 + \theta_4 X_1^2 + \cdots \geq 0$

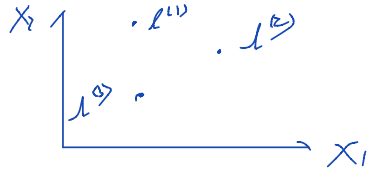
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \cdots \geq 0 \\ 0 & \text{else} \end{cases}$$

Can rewrite: $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

where: $f_1 = X_1$, $f_2 = X_2$, $f_3 = X_1 X_2$ ...

Q: How to optimize ~~the~~ choice of $f_1, f_2, f_3 \cdots$?

Kernel



pick landmarks $l^{(1)}, l^{(2)}, l^{(3)}$, given x, Compute new feature depend on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

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

$$f_2 = \text{Similarity}(x, l^{(2)}) = \exp(\cdots)$$

$$f_3 = -$$

Kernel $k(x, l^{(1)})$ Gaussian kernel

Kernels and Similarity

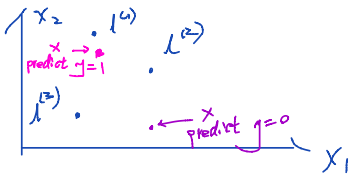
$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$$

if $x \approx l^{(1)}$: $f_1 \approx 1$

if x far from $l^{(1)}$, $f_1 \approx 0$

each of landmarks defines a new feature

Example:



predict 1 when $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 \geq 0$

say $\theta_0 = -0.5$, $\theta_1 = 1$, $\theta_2 = 1$, $\theta_3 = 0$

case 1: $f_1 \approx 1$, $f_2 \approx 0$, $f_3 \approx 0$

$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0 = -0.5 + 1 = 0.5 \geq 0$ predict 1

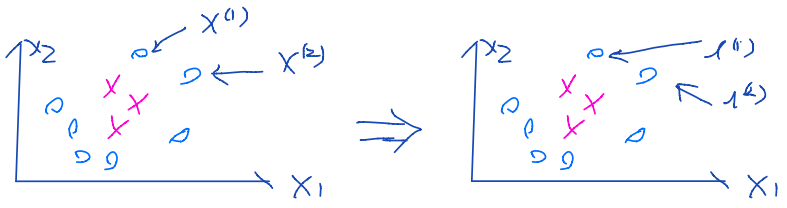
case 2: $f_1, f_2, f_3 \approx 0$

$\theta_0 + \theta_1 \times 0 + \theta_2 \times 0 + \theta_3 \times 0 = -0.5 < 0$ predict 0

Understand: for points close to landmarks, predict 1

for point far away from landmarks, predict 0

Choosing the landmarks



choose landmark exactly where the training data is

m training examples $\Rightarrow$ m landmarks

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

choose $l^{(1)} = x^{(1)}, \cdots\cdots l^{(m)} = x^{(m)}$

Given example x:

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

$f_2 = \cdots\cdots$

$f_m = \cdots\cdots$

$f = \begin{bmatrix} f_0 \rightarrow = 1 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$

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

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

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

$\vdots \qquad \vdots \longleftarrow f_i^{(i)} = \text{Sim}(x^{(i)}, l^{(i)}) = 1$

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

then: $x \in \mathbb{R}^{n+1} \rightarrow f^{(i)} = \begin{bmatrix} f_0^{(i)} = 1 \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$ new feature function

SVM with kernels ($n = m$ or $n = m+1$)

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

predict "$y = 1$" if $\theta^T f \geq 0$ $\quad \theta^T f = \theta_0 f_0 + \theta_1 f_1 + \cdots \theta_m f_m$ $\theta \in \mathbb{R}^{m+1}$

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

In real software/packages, it's actually $\frac{1}{2} \sum_{j=1}^{m} \theta^T M \theta$ instead of $\|\theta\|^2$

where M is some kernel related matrix, in order to let the algorithm scale to much large datasets and computes faster

* SVM with kernel usually runs slow

SVM parameters:

$C(=\frac{1}{\lambda})$: Large C: lower bias, high variance
Small C: higher bias, low variance

σ^2: large σ^2: features f_i vary more smoothly
higher bias, lower variance
Small σ^2: feature f_i vary less smoothly
lower bias, higher variance

Using An SVM

svm software package: liblinear, libsvm
Need to specify:
- choice of parameter C
- choice of kernel

e.g. No kernel ("linear kernel")
predict "y = 1" if $\theta^T x \geq 0$
- good for n (# of features) large, m (# of examples) small
to avoid overfitting

e.g. Gaussian kernel
need to choose σ^2
* good for n small, m large
** Note: Do preform feature scaling before using Gaussian kernel, reason is

$$\|x - l\|^2 = (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$$

then x_2 doesn't show much influence

other choice of kernel:

Note: Not all similarity functions make valid kernels

Need to satisfy "Mercer's Theorem" to make sure SVM packages' optimizations run correctly and do not diverge.

Many off-the-shelf kernels available:

- polynomial kernel: $(x^T l + \text{const})^{\text{degree}}$ (=2, 3, ……)
- string kernel
- chi-square kernel
- histogram intersection kernel

Multi-class classification

Many SVM packages already have built-in multi-class classification functionality

Otherwise, use one-vs.-all method. Train K SVMs, one to distinguish $y=i$ from the rest, for $i=1, 2, \cdots K$, get $\theta^{(1)}, \theta^{(2)}, \cdots \theta^{(K)}$, pick class i with largest $(\theta^{(i)})^T x$

Logistic regression VS. SVMs

How to make choice in between these two algorithms?

n: number of features ($x \in \mathbb{R}^{n+1}$)

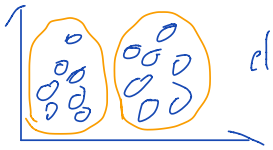
m: number of training examples

- n is large relative to m (e.g. n ≥ m, n = 10,000, n = 10, ... 1000)
  - ↳ Use regression, or SVM w/o a kernel ("linear kernel")
- n is small, m is intermediate (e.g. n = 1–1000, m = 10 – 10,000)
  - ↳ Use SVM w/ Gaussian kernel
- n is small, m is large (n = 1 ~ 1000, m = 50,000 +)
  - ↳ Create/add more features, then use logistic regression or SVM w/o a kernel ("linear kernel")
- neural network likely to work well for most of these settings, but may be slower to train

* SVM is a convex problem, can always find the global min

Clustering — Unsupervised Learning Intro

Unsupervised Learning



clustering algorithm

Training set $\{x^{(1)}, x^{(2)}, x^{(3)}, \cdots\cdots, x^{(m)}\}$

Application of clustering.

- Market segmentation
- Social Network analysis
- Organize Computing clusters
- Astronomical data analysis
- ……

clustering is an example of unsupervised learning

Clustering — K-means algorithm

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

    $x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)
- Procedure
  - Randomly initialize K cluster centroids $\mu_1, \mu_2, \cdots \mu_K \in \mathbb{R}^n$
  - Repeat {

    for $i = 1$ to m (cluster assignment)

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

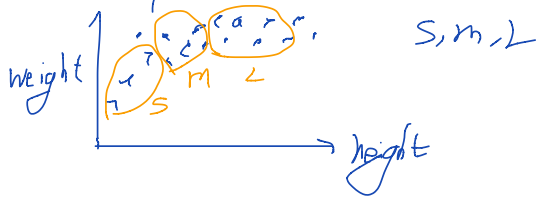
for k = 1 to K: (update centroid)

μ_k := average (mean) of points assigned to cluster k

}

If not points assigned to a certain centroid during a certain step, then elimate the corresponding centroid or randomly re-initialize that centroid

K-means for non-separated clusters



S, M, L

- clustering — optimization objective

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

Optimization objective

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

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

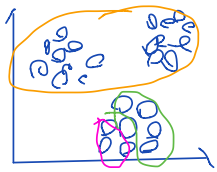
Cost function / distortion of K-mean algorithm

- clustering – Random initialization

  - should have $K < m$
  - Randomly pick K training examples
  - set $\mu_1, \dots \mu_K$ equal to these K examples

Problem of local optima

Assume $K=3$, could end up with:

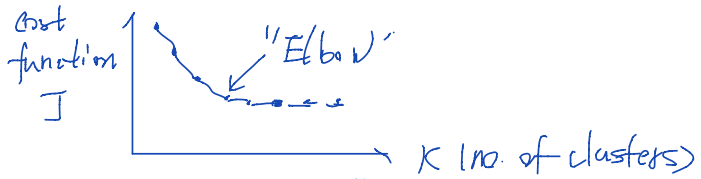


Solution: Run K-means clustering several times

```
for i=1 to 100 {        → 50 – 1000
    Randomly initialize K-means
    Run k-means, Get c^(1), ... c^(m), μ1, ... μK
    Compute cost function (distortion)
        J(c^(1), ... c^(m), μ1, ... μK)
}
Pick clustering that give lowest cost J(...)
```

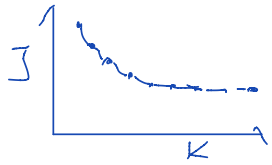
- Clustering – choosing the number of clusters

— Elbow method



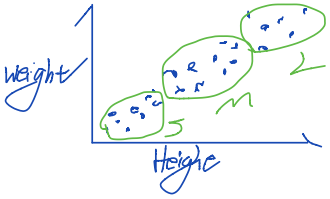
choose K as where corresponding to the "Elbow"

* but "Elbow" is not always obvious, e.g.

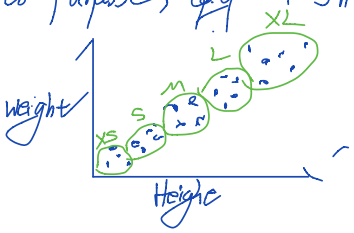


— Base on later purpose

Sometimes you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose, e.g. T-shirt size:



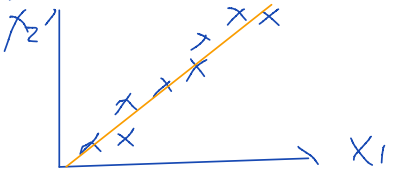
size: S, M, L



size: XS, S, M, L, XL

Dimensionality Reduction

- Motivation I: Data Compression

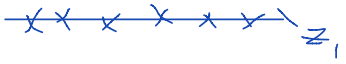


$x^{(1)} \in \mathbb{R}^{(2)} \rightarrow z^{(1)} \in \mathbb{R}$

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

$x^{(3)} \in \mathbb{R}^{(3)} \rightarrow z^{(3)} \in \mathbb{R}$

2D $\rightarrow$ 1D



* this is purely a data projection

3D → 2D: Project 3D data to 2D plane

- Motivation II: Data visualization

reduce dimentionality to view high dimention data on 2D or 3D or 1D

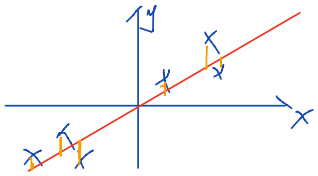
Principal Component Analysis (PCA)

- Problem formulation

Reduce 2D → 1D: find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) on to which to project the data so as to minimize the projection error.

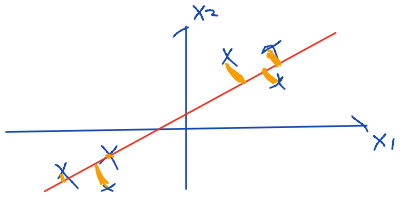
Reduce from n-D → k-D: find k vectors $u^{(1)}, \dots, u^{(k)}$ on to which to project the data, so as to minimize the projection error.

* PCA is not Linear Regration



Linear regression

{ x Vs y



PCA

{ x_1 Vs x_2, no y

vertical distance projection distance

- PCA Algorithm

1) Data preprocessing (feature scaling / mean normalization)

- $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$
- Replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$
- if different features on different scales, scale features to have comparable ranges

Problem: how to compute $\mu^{(i)}$ and z

2) Algorithm: reducing data from $n \to k$:

- Compute "Covariance matrix"

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

- Compute "eigenvectors of matrix Σ":

[U, S, V] = svd(sigma) (In mathematica)

svd: Singular Value decomposition

$$U (n \times n) = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix}}_{K} \text{ eigenvectors}$$

Take the first k eigenvectors

- Take first k eigenvectors ... - $\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \end{bmatrix}$

U_{reduce} — $n \times k$

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

PCA actually minimizes squared projection error

- Reconstruction from compressed representation

  How do you go back from $z^{(i)}$ to $x^{(i)}$?

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

$$z = U_{reduce}^T x \Rightarrow x_{approx}^{(i)} = \underset{n \times 1}{U_{reduce}} \cdot \underset{k \times 1}{z^{(i)}}$$

- choosing the number of principal Component k

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

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

Typically, choose k to be smallest value so that:

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

("99% of variance is retained")

Algorithm:

try $k = 1, 2, \ldots$

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

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

if true, break. find good k value

if False, continue.

$[U, S, V] = \text{Svd}(\text{Sigma})$

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

for a given k:

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.1$$

Only need to update numerator for different k

- Advice for applying PCA

Use PCA to speed up learning algorithm

Supervised learning speed up

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

Extract inputs:

unlabled dataset: $x^{(1)}, x^{(2)}, \ldots x^{(m)} \in \mathbb{R}^n$

$\Downarrow$ PCA

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

New training set:

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

New test example

$x \xrightarrow{\text{PCA}} z$

$y = h_\theta(z) \longrightarrow$ predictions

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x_{cv}^{(i)}$ and $x_{test}^{(i)}$ in the cross validation and test sets

Application of PCA:

- Reduce memory/disk needed to store data
- Speed up learn algorithm
  (choose K by % of variance retain)
- Visualization
  K=2 or K=3 for visualize high dimentional data

Bad use of PCA I:

Bad thinking: PCA → reduce num of features
→ prevent overfitting

Actually: This might be ok, but isn't a good way to address overfitting. Use regularization instead

Because: PCA doesn't information of y involved, it might through away some valuable information.
While regularization has all information retained.

Bad use of PCA II.

Design of ML system (Bad)

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

2) Run PCA to reduce $x^{(i)}$ to $z^{(i)}$

3) Train

4) Test on test set: map $x_{test}^{(i)}$ to $z_{test}^{(i)}$, run $h(z^{(i)})$

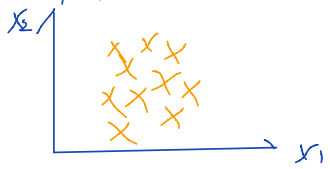
Suggestion:

Before implementing PCA, first ~~try~~ running ML on ~~the~~ raw data, Only if ~~that~~ doesn't do what you want, ~~then~~ implement PCA

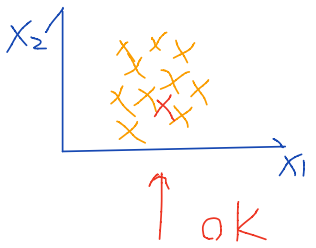
Anomaly Detection — Problem motivation

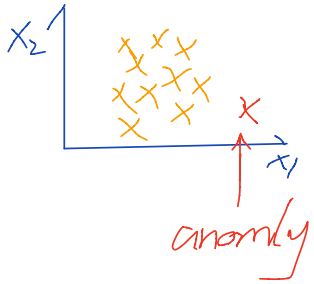
- An Example

Features $\{x_1^{(1)}, x_2^{(1)}\}, \cdots\cdots, \{x_1^{(m)}, x_2^{(m)}\}$ has distribution:



If now having a new data set, it could be of following cases:





- Density Estimation

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

Is x_{test} anomalous?

Model: density $p(x)$ $\begin{cases} p(x_{test}) < \varepsilon & \rightarrow \text{anomaly} \\ p(x_{test}) \geq \varepsilon & \rightarrow \text{ok} \end{cases}$

Example:

— Fraud detection

- $x^{(i)}$ features of user i's activities
- Model $p(x)$ from data

- Identify unusual users by checking which have $p(x) < \varepsilon$

— Manufacturing

— Monitoring computers in a data center

Anomaly Detection — Gaussian distribution

Anomaly Detection — Algorithm

◦ Density Estimation

Training set $\{x^{(1)}, \cdots x^{(m)}\}$ $\quad x \in \mathbb{R}^n$

$p(x) = P(x_1; \mu_1, \sigma_1^2) P(x_2; \mu_2, \sigma_2^2) \cdots P(x_n; \mu_n, \sigma_n^2)$

Assume each feature follows Gaussian distribution, and they are independent to each other. (In computation, even if they are not independent, it's fine ?? — Andrew said so, I'm not sold)

$$p(x) = P(x_1; \mu_1, \sigma_1^2) \cdots P(x_n; \mu_n, \sigma_n^2)$$
$$= \prod_{j=1}^{n} P(x_j; \mu_j, \sigma_j^2)$$

Algorithm:

1. Choose features x_i that you think might be indicative of anomalous examples
2. Fit parameters $\mu_1, \cdots \mu_n, \sigma_1^2, \cdots \sigma_n^2$

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

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

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

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

anomaly if $p(x) < \varepsilon$

Building an anomaly detection system

- Develop anomaly system

The importance of real-number evaluation

When developing a learning algorithm, making decisions is much easier is we have a way of evaluating our algorithm

Assume we have some labeled data, of anomalous and non-anomalous examples ($y=0$ if normal, $y=1$ if anomalous)

Training set: $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$, assume they are all normal examples (it's ok if a few anomalous examples are included.)

Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \cdots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

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

are labeled data sets (meaning we know exactly if they are anomalous or non-anomalous

Aircraft engines motivating example:

10000 good (normal) engines, 20 flawed engines (anomalous)

Training set: 6000 good engines ($y=0$), 10 anomalous ($y=1$)
CV set: 2000 good engines ($y=0$), 10 anomalous ($y=1$)
Test set: 2000 good engines ($y=0$), 10 anomalous ($y=1$)

Algorithm:

1) Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$
2) On a cv/test set example x, predict

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

Possible evaluation metrics:

- True positive, false positive, false negative
  true negative
- Precision / Recall
- F_1-Score

Can also use CV set to choose ε value

Note CV/test set is very skewed!! So classification accuracy is not a good way to evaluate the algorithm.

- Anomaly detection VS supervised learning

Anomaly detection	Supervised learning
• very small num of positive examples (y=1)	• Large number of positive and negative examples
• large num of negative examples (y=0)	• Enough positive examples for algorithm to get sence of what positive examples are look like
• Many different "types" of anomaly examples hard for any algorithm to learn from it	• Future positive examples likely to be similar to ones in training set
• future anomalies may look nothing like any of the anomalous examples seen before	

Application

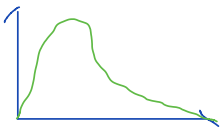
Anomaly detection	Supervised learning
• Fraud dection	• Email spam classification
• manufacturing	• Weather prediction
• Monitoring machines in data center	• cancer classification
⋮	⋮

If have large number of positive examples (y=1), anomaly detection can be transfered to supervised learning

o choosing what features to use

non-gaussian features transform

- Plot the feature histogram to exam if it's Gaussian
- If its not a good gaussian



do some transformation to the feature

to make it more gaussian, e.g. $\log(x+c)$, $\sqrt{x}$, x^t, ...

Error analysis for anomaly detection

- want: $p(x)$ large for normal ($y=0$) examples x

  $p(x)$ small for anomalous ($y=1$) examples x.

- most common problem:

  $p(x)$ is comparable for normal/anomalous examples

- Solution

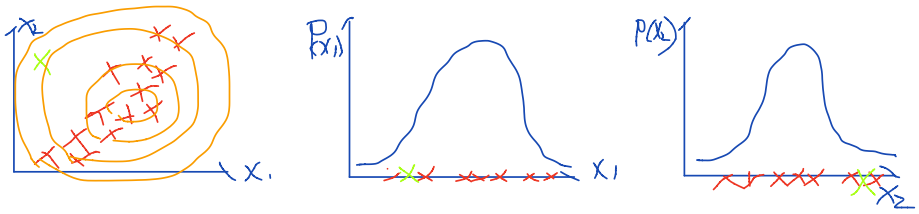
  Inspect mistakes and try to create new features

- experience

  · choose features that might take on unusually large or small values in an event of an anomaly

Multivariate Gaussian Distribution

- When anomaly detection fails:



Fail to detect x as positive

Solution:

$x \in \mathbb{R}^n$, don't model $p(x_1), \cdots, p(x_n)$ seperately

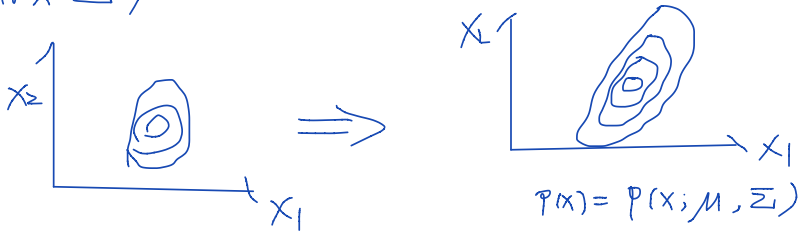
Model $p(x)$ all in one go

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

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

$|\Sigma| = \det(\Sigma)$

By doing this, actually we're allowed to "rotate" the Gaussian distribution (due to off-diagnal value of covariance matrix Σ)



$p(x) = p(x_1)p(x_2)\cdots p(x_n)$

$p(x) = p(x; \mu, \Sigma)$

- Algorithm

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

Recommender Systems

- Problem formulation

Example: predicting movie ratings

MOVIE	USER 1	USER 2	USER 3	USER 4	.....
movie 1	5	?	3	4	
movie 2	1	2	?	4	
⋮	..	..	..	..	

n_u = no. users n_m = no. movies

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

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

Question: how to predict movie ratings that hasn't been rated by a certain user?

- Content based recommendations

$n_u = 4$, $n_m = 5$

x_1 (romance) x_2 (action)

MOVIE	USER 1	USER 2	USER 3	USER 4		
movie 1	5	?	3	4	0.9	0
movie 2	1	2	?	4	1.0	0.01
⋮	⋮	⋮	⋮	⋮	⋮	⋮

$x_0 = 1 \quad x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix} \quad x^{(2)} = \begin{bmatrix} 1 \\ 1.0 \\ 0.01 \end{bmatrix} \cdots\cdots$

num of features : 3 $x_0 = 1$ x_1, x_2

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

Problem formulation:

- $r(i,j) = 1$ if user j has rated movie i, else 0
- $y^{(i,j)}$ = rating by user j on movie i (if defined)
- $\theta^{(j)}$ parameter vector for user j
- $x^{(i)}$ feature vector for movie i
- for user j movie i, predicted rating $(\theta^{(j)})^T x^{(i)}$

To learn $\theta^{(j)}$, $m^{(j)}$ = no. of movies rated by user j

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

just like linear regression, use gradient decent to find θ

Collaborative Filtering (feature learning)

MOVIE	USER 1	USER 2	USER 3	USER 4	x_1	x_2
movie 1	5	?	3	4	?	?
movie 2	1	2	?	4	?	?
⋮	⋮	⋮	⋮	⋮	⋮	⋮
					?	?

If ~~the~~ users can ~~tell~~ us how much ~~they~~ like each kind of movies (e.g. $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$, $\theta^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$ ...)
~~then~~ we can learn ~~the~~ features for each rated movies

The problem turned out ~~to~~ be; what feature value $x_1^{(i)}$ for movie i should be, so that $(\theta^{(1)})^T x^{(i)} \approx 5$, $(\theta^{(2)})^T x^{(i)} \approx 0$, ...

Optimization algorithm

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

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

Collaborative filtering

- Given $x^{(1)}, \dots, x^{(n_m)}$ (and movie ratings)
  can estimate $\theta^{(1)}, \dots, \theta^{(n_u)}$
- Given $\theta^{(1)}, \dots, \theta^{(n_u)}$

can estimate $x^{(1)}, \dots, x^{(n_m)}$

$\Rightarrow$ Guess $\theta \to x \to \theta \to x \to \theta \to x \to \theta \to \cdots$

This is awesome !!!

- Collaborative filtering algorithm

non-optimized algorithm

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

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

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

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

Guess $x^{(1)}, \dots x^{(n_m)}$ or $\theta^{(1)}, \dots \theta^{(n_u)}$, then go back and forth (i) and (ii) to optimize the learning

— Optimized algorithm

Do (i) and (ii) at the same time

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

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

no $x^0 = 1$ in this case

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

2) Minimize $J(x^{(1)} \dots, \theta^{(1)} \dots)$ using gradient decent, e.g.

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

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

3) for a user with parameters θ and a movie with features x, predict a star rating of $\theta^T x$

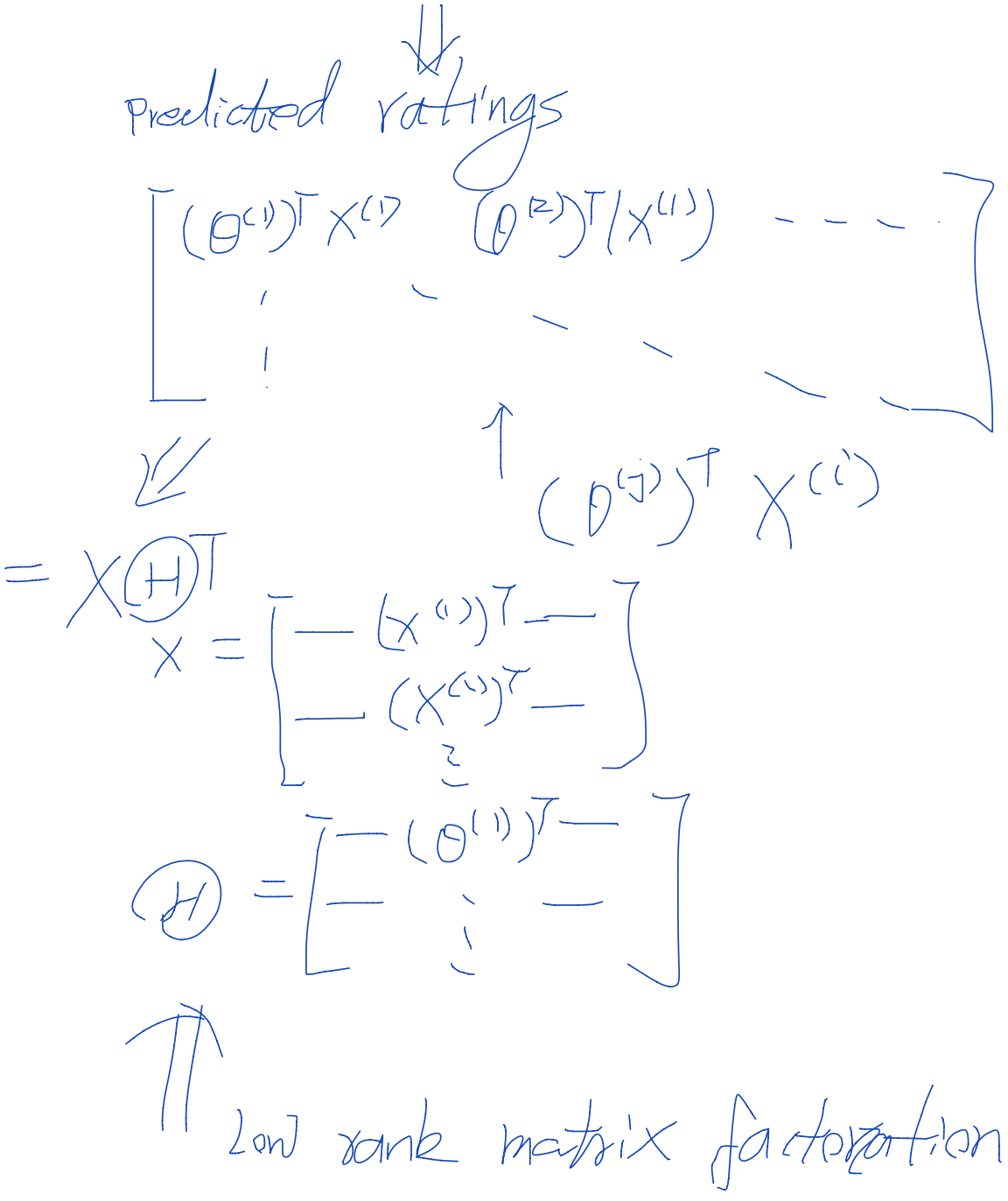
Low rank Matrix Factorization

MOVIE	USER 1	USER 2	USER 3	USER 4	.....
movie 1	5	?	3	4	
movie 2	1	2	?	4	
⋮	⋮	⋮	⋮	⋮	

$$Y = \begin{bmatrix} 5 & ? & 3 & 4 \\ 1 & 2 & ? & 4 \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

$y(i,j)$

1.



- Finding related movies
  - For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$
  - ? How to find movies j related to movie i ?
    Small $\|x^{(i)} - x^{(j)}\|$ → movie i, i are "similar"

Implementational detail: Mean Normalization

For users hasn't rated any movie, $\min J(\theta) \longrightarrow$ with give $\theta = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ so predict this user will always rate movie 0

NOT RIGHT!

Solution: Mean Normalization

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

$$\Rightarrow \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ \vdots \\ \end{bmatrix} \longrightarrow Y' = \begin{bmatrix} 2.5 & 2.5 & -2.5 & 2.5 & ? \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \end{bmatrix}$$

For user j on movie i, predict $(\theta^{(j)})^T(x^{(i)}) + \mu_i$

In this case the above problem can be avoided.

Learning with large datasets

Stochastic Gradient Descent

- Linear regression with gradient decent

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

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

Repeat {

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

$j = 0, 1, \ldots, n$

}

$\boxed{\quad}$: Batch gradient decent

Computational expensive since need to load all m examples to do one small step

scanning ~~through~~ the entire dataset peridically

- Stochastic gradient decent

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

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

1) Randomly shuffle dataset

2) Repeat { (usually 1 ~ 10 times repeating, depends on m)

for $i = 1, \ldots, m$ {

$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

(for $j = 0, 1, \ldots, n$) $= \frac{\partial}{\partial \theta_j} cost(\theta, x^{(i)}, y^{(i)})$

}

}

Why stochastic gradient descent is much faster than gradient descent?

- for large dataset, gradient decent has to run through all training set to make on one baby step of descent
- Stochastic gradient dexcent take one descent step only when run through one training example. Especially when dataset is large, running through all training examples once is enough to get good training result.

- Mini-batch gradient descent

Batch Gradient Descent uses all m

Stochastic Gradient Descent uses 1

Mini Gradient Descent uses b

} Examples in each iteration

- Mini-batch gradient descent

1) choose b.

2) Repeat {

for $i = 1, 1+b, 1+2b, \cdots$ {

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

(for every $j = 0, 1, \cdots n$)

}

}

- Mini-batch gradient descent can be faster than stochastic gradient descent
  - ☆ vectorization!
  - Mini-gradient descent use b examples for one step, can be vectorized, meaning partially parallelize !!!

Stochastic Gradient Descent Convergence

- Checking for convergence
  - $\text{Cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})$
  - During the learning compute cost $(\theta, (x^{(i)}, y^{(i)}))$ before updating using $(x^{(i)}, y^{(i)})$
  - Every 1000 iterations (say), plot cost averaged over the last 1000 examples processed by the algorithm.

* Stochastic gradient descent does not converge to the global minimum, but only oscillates around the global minimum

* Still need to choose the proper learning rate α.

* if want to SGD to converge, can slowly decrease learning rate α over time

  e.g. $\alpha = \frac{\text{const 1}}{\text{iteration\# + const 2}}$

Online learning

Eg1: Shipping service website. users sometimes choose your shipping service ($y=1$)
Sometimes not ($y=0$)
Features x capture properties of users, want to learn $P(y=1|x;\theta)$
to optimize the prize to get maximum users

Background: have large stream of user's data

Repeat {
    Get (x,y) corresponding to user
    update θ using (x,y)
        $\theta_j := \theta_j - \alpha(h_\theta(x)-y)\cdot x_j$ $(j=0,1,\ldots,n)$
    Discard (x,y) after using it once
}

* Advantage of Online learning
    - Can adapt to change of users' preferences

Eg 2: Product Search (learning to search)

User searches for "Android phone 1080p camera"
Have 100 phones in store. Will return 10 results
x: features of phone that matches the query

y: 1 if users clicked ~~the~~ button

0 ~~otherwise~~

learn $P(y=1|x;\theta)$ [predicted CTR]

CTR: Click through rate

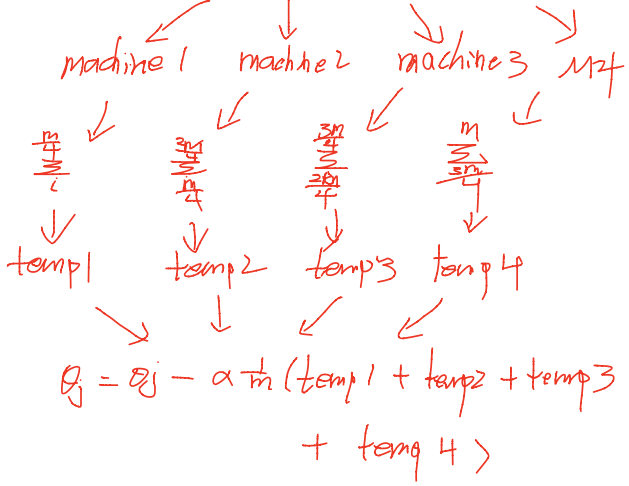
other examples: choosing special offers ~~to~~ users; customized of selection of news articles; product recommendation

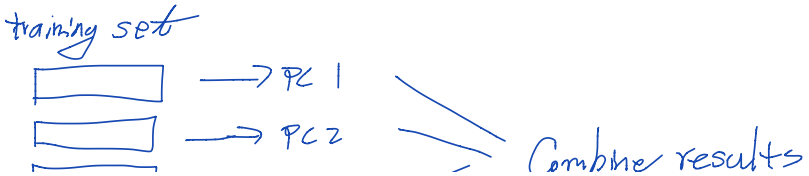
These examples can also be ~~transfer to~~ classical ML problem

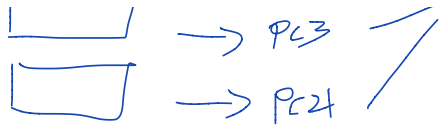
Map-reduce and data parallism

- Map-reduce

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







Many learning algorithms can be expressed as computing sums of functions over ~~the~~ training set

- Multi-cores machines

  pc i => core i

Photo OCR

- Problem description and pipeline

  photo OCR : photo Optical Charccter Recognition

- Photo OCR Pipeline

  1. Text detection
  2. character segmentation
  3. character classification

  Each above steps may be a machine learning problem

  Pipeline : A system with many components / stages, several of which may use machine learning

- sliding windows

  Text detection.

Getting lots of data : Artificial data synthesis

- Creating new data from scratch.

  Take real data. and add background / noise / distortion

  Distortion should represent ~~the~~ type of noise in ~~the test set~~.

- Discussion on getting more data.

  1. Make sure you have low bias classifier before expaneling ~~the effort~~
  2. How much work would it be to ~~get~~ 10X much data as we

currently have?

- Artificial data synthesis?
- Collect/label it yourself?
- Crowd source (e.g. Amazon mechanical Turk

Ceiling Analysis: What part of pipeline to work on next?

Image → Text detection → character segmentation → character recognition

Ceiling analysis: estimating the error due to each component