Github: JaneShi99

Coursera Machine learning course
Notes By Jane Shi

Estimated completion: Dec 2018 - Jan 2019.

Week 1

Introduction

Definition of Machine learning.

A computer program is said to learn from experience E with respect to some task T and some performance measure P if its performance on T as measured by P improves with experience E.

Main two types: (explored in this course)

↳ Supervised machine learning vs unsupervised machine learning

↳ advice for practical uses of machine learning

↳ how to develop ML systems?

Supervised learning

↳ we gave a dataset (where the "right answer" are given) we know what our answer looks like as a result of relation between input & output

↳ regression problem: predict continuous value output.

↳ classification problem: predict discrete value output.

↳ take account of various number of inputs. / features / infinite many ~~the~~ attributes

Unsupervised learning

↳ determine clustering of data, where we have little / no idea about what result should look like

↳ identify cohesive groups of data

↳ example: cocktail party problem.

given two recording, with two tracks mixed at different volume, output each sound track

↳ can be written in one line (solution).

↳ Octave is good! built for lin. alg. & related programming.

Model & cost function

Model representation

↳ linear regression model

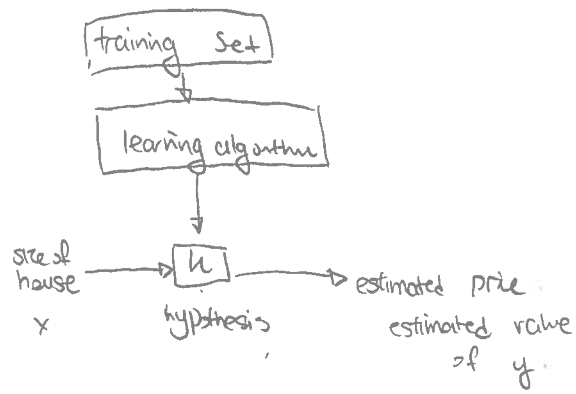
↳ training set is the data-set.

↳ m = # of training example, x's input var/feature, y's output var

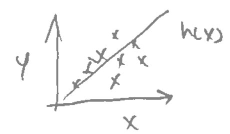
(x, y) ← training example

$(x^{(i)}, y^{(i)})$ - ith training example.

↳



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



↳ this is lin. reg w/ 1 variable / univariate lin. reg.

Cost function

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

↑ ↑ params

↳ goal is to minimize $\frac{1}{2m}\sum_{i=1}^{m}(\underbrace{h_\theta(x^{(i)})}_{= \theta_0 + \theta_1 x^{(i)}} - y^{(i)})^2$ ← # of training examples

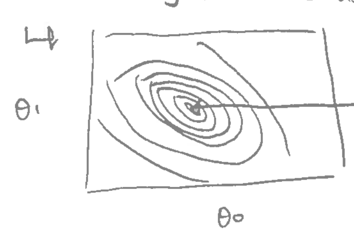
↳ minimize$_{\theta_0, \theta_1}$ $J(\theta_0, \theta_1)$ where $\underbrace{J(\theta_0, \theta_1)}_{\text{cost function.}} = \frac{1}{2m}\sum (h_\theta(x^{(i)}) - y^{(i)})^2$

↳ the square error function.

↳ $J(\theta_0, \theta_1)$ is a function in θ_0, θ_1. Plot $J(\theta_0, \theta_1)$ & find your global minima

↳ contour graphs are used for multiple features. (plot 3D graph)

↳



global minima so that θ_0, θ_1 → line of best fit.

↳ the graphs cannot <u>always</u> be visualized as easily. Thus, we would need some other algo.

↳ Gradient descent algorithm.

↳ have function $J(\theta_0, \theta_1)$

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

↳ you start with some θ_0, θ_1, then keep changing $\theta_0, \theta_1 \rightarrow$ reduce $J(\theta_0, \theta_1)$ each iteration,

↳ via calculus

↳ you can ~~end~~ end up at two different local optimums

the algorithm: ($:=$) ← assignment operator

repeat until converge {

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

}

Correct <u>simultaneous</u> update:

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

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

$\theta_0 = temp0$

$\theta_1 = temp1$

↳ α is the learning rate.
how big step we go down hill?
big step / baby step?

↳ simultaneously update θ_0 and θ_1 at same time.

↳ when updating, takes consideration of whether $\frac{\partial}{\partial \theta_0}$ is positive or negative, so the new point is closer to x axis. (the absolute value of $\frac{\partial}{\partial \theta_0}$ approach to 0 gradually).

↳ need to chose α so it's not too small, not too large.

if α too small → slow algorithm

if α too large, → may even diverge

↳ Putting it altogether:

Gradient Descent algorithm

repeat until converge {

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

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

}

linear regression model.

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

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

apply ↓ to ← to minimize

Plug in the equation, we obtain

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

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

θ_0 : $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$

θ_1 : $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$

It's always a convex function.
Our linear regression algorithm turns out to be

repeat until converge {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

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

}.

"Batch Gradient Descent" each step of gradient descent uses all training examples
↳ note: must use the model for $J(\theta_0, \theta_1)$ where there's no other local optima than the global.
or else it can end up at another local min

$\boxed{\text{Week 2}}$

Multi-feature linear regression
↳ having multiple features
notation: n = # of feature
$x^{(i)}$: input features of i^{th} example (vector)
$x_j^{(i)}$: value of feature j in the i^{th} training example.

↳ hypothesis:

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

for convenience, $\forall x$, $x_0 = 1$

so $h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i$

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

Hypothesis:

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

or inner product, $\langle \theta, x \rangle$

Parameter: θ

Cost function:

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

Gradient descent:

repeat {

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

}

simultaneous update θ_j, $j = 0, 1, 2, \cdots n$. since you're taking derivative with respect to ith feature.

Gradient descent in practice.

↳ feature scaling.

↳ make sure features are on a similar scale.

↳ $\forall i$, $-1 \le x_i \le 1$.

↳ major values around $-3 \sim +3$ ish not too little as in ~ 0.1

↳ Mean normalization

↳ replace x_i with $x_i - \mu_i$, to make sure feature have ~ 0 mean

↳ do not apply to $x_0 = 1$ though!

$$x_i \leftarrow \frac{x_i - \overset{\text{average value of } x_i}{\mu_i}}{s_i} \leftarrow \text{range or std.}$$

↳ "debugging" make sure it works properly.

↳ how to choose your α?

↳ "Debugging" make plot where # iter is x-axis, min $J(\theta)$ y,

↳ $J(\theta)$ should always decrease due to # of iter (every single iter!)

↳ if $J(\theta)$ error increases, you want to decrease α.

↳ Convergence test: choose ε to declare when $J(\theta) < \varepsilon$. → Converges!

↳ tip: to choose α, try 0.001, 0.01, 0.1, 1, ... try a range of values.

0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, ...

Features & polynomial regression

↳ Combine multiple features into 1.

↳ combine x_1 and x_2, by taking $x_3 = x_1 \cdot x_2$

↳ polynomial regression if linear doesn't fit.

↳ change the behaviour, so it can be quadratic / cubic etc.

↳ ideas: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

feature x_2 feature x_3

$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x_1}$

↳ with this though, keep in mind, feature scaling is very important.

Normal equation (computing param analytically)

↳ X: design matrix. $x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$, then $X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ \vdots \\ -(x^{(m)})^T- \end{bmatrix}$

↳ optimum θ given by

$\theta = (X^T X)^{-1} X^T y$ octave: print (x' * x) * x' * y x' : transpose * : matrix mult

↳ note with normal equation, you DON'T need feature scaling.

Gradient descent vs Normal equation

{ m # training example
{ n # of feature

Gradient descent:
↳ need to choose α
↳ many iteration
↳ work well even if n is large.

Normal equation:
↳ no need to choose α
↳ no iteration needed
↳ $(X^T X)^{-1}$ takes $O(n^3)$
↳ slow when n is large ($\geq 10{,}000$)

Normal equation / noninvertibility

↳ what if X^TX is non-invertible?

↳ use 'pinv' instead of 'inv' (pseudo-inverse)

↳ it gives you θ though X^TX is singular

↳ ① happen when there's redundant feature. or ② too many feature: m<n, then use regularization) / or delete features,

Vectorization

helps to compute vectors faster.

Assignment questions include:

↳ computing cost for multi/uni variable dataset

↳ computing cost for multiple variable

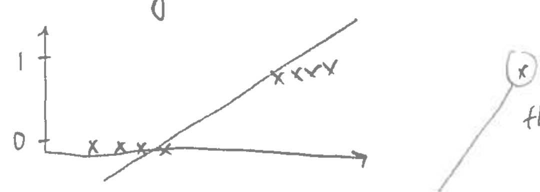
↳ gradient descent for multi/uni variables.

$\boxed{\text{Week 3}}$

Work with classification problem

↳ $y \in \{0, 1\}$ 0 Negative class, 1 Positive class.

↳ now: binary class classification. Does lin-reg work? no. not a good idea.



threshold = 0.5.

If $h_\theta(x) \begin{cases} \geq th \\ < th \end{cases}$ predict $\begin{cases} 1 \\ 0 \end{cases}$

↳

this is the bug! mess up your lin reg data! :(

↳ So don't use lin reg for classification.

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

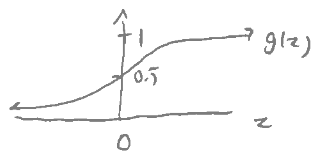
this is a classification algorithm

logistic regression

↳ want: $0 \le h_\theta(x) \le 1$

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

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



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

↳ Interpretation: $h_\theta(x)$ gives you probability that our output is 1. $= P(y=1 \mid x;\theta)$ in probability notation.

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

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

Decision Boundary

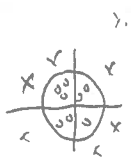
↳ $h_\theta(x) \geq 0.5 \rightarrow y=1$ or $\theta^T x \geq 0 \rightarrow y=1$

$h_\theta(x) < 0.5 \rightarrow y=0$ or $\theta^T x < 0 \rightarrow y=0$

since ↳ $g(z) \geq 0.5 \Leftrightarrow z \geq 0$.

decision boundary is the line that separate area when $y=0, y=1$ (line where $h_\theta(x) = 0.5$ exactly.)

there are also non. linear decision boundaries, then you need more params for higher dim. ie.



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

$\rightarrow -1 + x_1^2 + x_2^2 \geq 0$ results in ○ boundary

logistic regression model

training set $= \{(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})\}$

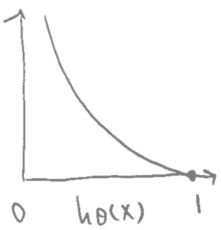
m examples, $x = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ $x_0 = 1$, $y \in \{0,1\}$

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

lin reg won't give a convex, but we want a convex function

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

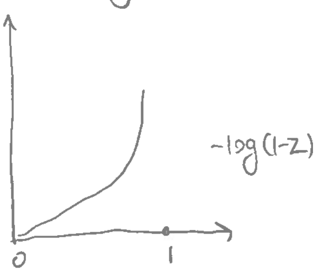
if $y=1$



cost=0 if $x=1$ but as $h_\theta(x) \to 0$, cost $\to \infty$.

intuition: if $h_\theta(x)=0$, but you predict it as 1, you're penalized.

if $y=0$



similar as the other intuition

this gives a convex & local optimum free function

note: $y=1$ or $y=0$ always. → can combine two equations

the compressed cost function is:

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

total cost J:

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

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

want: $\min_\theta J(\theta)$

gradient descent algorithm:

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

or

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

looks same as lin.reg's grad.des, but! $h_\theta(x)$ refers to $\frac{1}{1+e^{-\theta^T x}}$ now.

vectorized implementation

cost

$h = g(X\theta)$ this computes quantity $h_\theta(x^{(i)})$

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

$\hookrightarrow J(\theta) = \frac{1}{m} \cdot (-y^T \log(h) - (1-y)^T \log(1-h))$

the gradient decent

Idea: rearrange the vectors until it's easy to type into Matlab. :)

$\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} [(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}]$ ← column vector

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

since: thus, X^T makes each $x^{(i)}$ a column vector.

$[\;]$ do column wise calculation.

θ is a $(n \times 1)$ vector.

$X\theta$ returns $x^{(i)}\theta \rightarrow$ returns corresponding index.

$$X: \quad x^{(i)} \begin{array}{c} \overbrace{\hphantom{xxxxxxxxxx}}^{n+1} \\ \begin{bmatrix} - & x^{(1)T} & - \\ & \vdots & \\ - & x^{(m)T} & - \end{bmatrix} \end{array} \Big\} m$$

θ: $n \times 1$ vec.

$X = m \times n$ $\quad X\theta = m \times 1$.

$X^T = n \times m$

ans: $n \times 1$.

Advanced Optimization

cost function $J(\theta)$, want $\min_\theta J(\theta)$.

Given θ, if we can compute $J(\theta)$, $\frac{\partial}{\partial \theta_j} J(\theta)$ then we can use the following algorithms

as long as you know these two you can use the lib functions

↳ Conjugate gradient

↳ BFGS

↳ L-BFGS

faster, no need for α, but more complex. So we use the library

use function "fminunc()"

Plugin the $J(\theta)$ & the gradients shall suffice

logistic optimization for multiple classes "one vs all classification"

Multiclass classification.

$y = \{0, 1, \cdots n\}$ each are category.

assign one class as positive, all other ones as "the rest"

$y \in \{0, 1, 2, \cdots n\}$

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

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

$\vdots$

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

Prediction: $\max_i h_\theta^{(i)}(x)$

Problem of over-fitting

under-fitting : hypothesis function maps too poorly to the trend of data. function too simple / too little features.

overfitting : not generalized enough. fits available data too well, but might have unnecessary angles/corners i.e. too wiggly (fail to generalized):

to resolve overfitting :
1) reduce # of features. (model selection algorithm to ditch less-important features.)
2) regularization (reduce magnitude of θ_j)

Cost function (the new one with regularization)

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

big λ bumps up and forces θ_j to be small because big θ_j will be penalized

Gradient descent

repeat {

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

$$\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \quad j \in \{1, 2, \ldots n\}$$

}

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

always less than 1
as it reduce θ_j each time
by a little bit.

Normal equation

$$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y$$

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

note if $m < n$, $X^T X$ is noninvertible but adding L makes it invertible.
regularization solves non invertibility as well.

Regularized logistic regression (advanced optimization works similarly).

regularized cost function for linear regression.

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

new term

Gradient descent :

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

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

Advanced functions (regularization).

Jval: same as previous.

gradient 1 = (index 0)

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

gradient (2 ~ n+1) index (1, 2, ... n))

$$\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_1^{(i)}\right)+\frac{\lambda}{m}\theta_j$$ ↙ newly added term

watch out following when doing assignment:

↳ display dimensions might be in opposite order.

↳ draw out matrices carefully to visualize the vectorization.

↳ match matrix dimensions always.

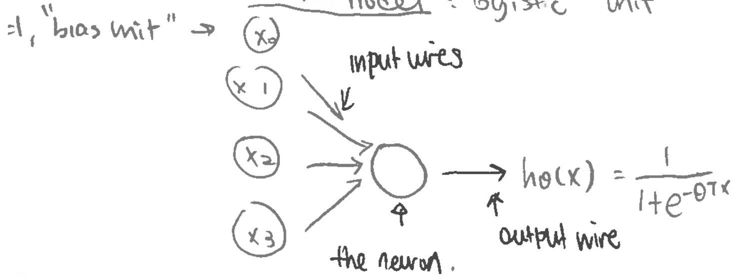
Week 4

Neural Networks - representation

- computer vision - example
  - logistic regression would have too many features. (like a few million ! for images)
- mimic the brain.
- large scale !
- neural-rewiring experiment
- adjust / learn the data

Model representation

Neuron model : logistic unit

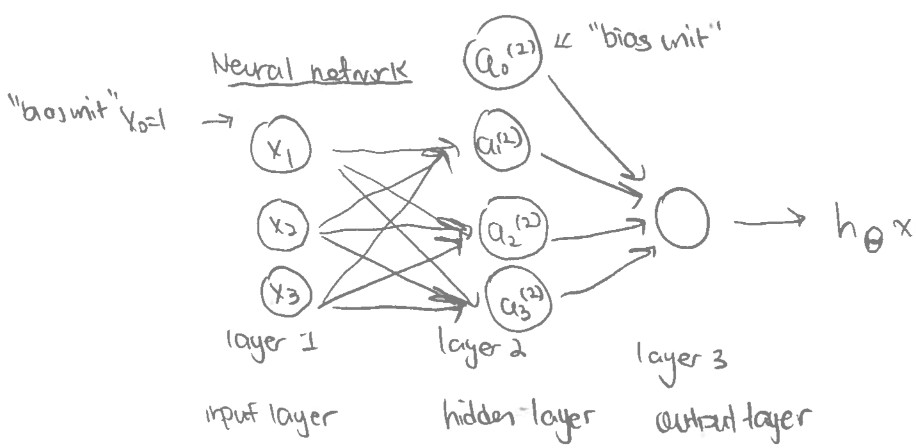


$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

" sigmoid activation function."

"weights"
"
"params"

Neural network



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

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

Vec representation

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \rightarrow h_\theta(x)$$

vector representation "activation nodes" example

layer 1 to layer 2:

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

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

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

layer 2 to layer 3

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

dim (Θ)

↳ each layer has its own matrix of weights

↳ if network has s_j layers in level j, s_{j+1} layers in level $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j + 1)$

comes from bias node

↳ laid out like this, b/c multiply Θ. the vector will be on the right.

intuition: Neural network allows nodes in its hidden layer to "learn" its own features.

Vectorization of computation

↳ $a_1^{(2)} = g(z_1^{(2)})$

↳ $a_2^{(2)} = g(z_2^{(2)})$

↳ $a_3^{(2)} = g(z_3^{(2)})$

refer here

for layer j, node k, z is

$$z_k^{(j)} = \Theta_{k,0}^{(j-1)} x_0 + \Theta_{k,1}^{(j-1)} x_1 + \dots + \Theta_{kn}^{(j)} x_n$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \vdots \\ z_n^{(j)} \end{bmatrix}$$

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

note: $\dim(\Theta^{(j-1)})$ is $s_j \times (n+1)$

$\dim(a^{(j-1)})$ is $(n+1) \times 1$.

$$a^{(j)} = g(z^{(j)})$$

adding the bias unit: to layer j after computing $a^{(j)}$ ie. $a_0^{(j)} = 1$

to compute final hypothesis, compute z vector:

$z^{(j+1)} = \Theta^{(j)} a^{(j)}$ the last matrix $\Theta^{(j)}$ has only 1 row, multiplied by one column vec $a^{(j)}$ so the result is a real number.

$$h_\Theta(x) = a^{(j+1)} = g(z^{(j+1)})$$

Multi-class Classification

↳ one-vs- all method

$h_\theta(x) \in \mathbb{R}^4$ if there are 4 classes.

$h_\theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ or $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ or $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ or $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

with different input x.

It returns one of the e_i's vector given a particular input

$\boxed{\text{Week 5}}$ Goal: learn how to train neural networks.

the cost function for the neural network.

↳ L = total # of layers in the network.

↳ s_l (# of units not counting bias unit in layer l)

↳ K = # of output unit / classes.

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

Back propagation algorithm

↳ goal is to compute $\min_\Theta J(\Theta)$

↳ look at partial derivative of $J(\Theta)$

$$\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta)$$

the back propagation algorithm works as follows:

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

↳ set $\Delta_{i,j}^{(l)} := 0 \quad \forall i,j$

↳ for training example $t = 1 \sim m$

1. set $a^{(1)} := x^{(t)}$
2. perform forward propagation to compute $a^{(l)}$, $l = 1, 2, 3, \cdots L$

   (i.e. set up z (intermediate), use $g(z)$ to calculate next layer)
3. $\delta^{(L)} = a^{(L)} - y^{(t)}$
4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \cdots \delta^{(2)}$ using $\delta^{(l)} = \underbrace{((\Theta^{(l)})^T \delta^{(l+1)})}_{\text{calc value}} .* \underbrace{a^{(l)} .* (1-a^{(l)})}_{g'(z^{(l)})}$

   $g'(z^{(l)}) = a^{(l)} .* (1-a^{(l)})$
5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_j^{(l+1)}$ with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

update: $D_{ij}^{(l)} := \begin{cases} \frac{1}{m}\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}, & \text{if } j \neq 0. \\ \frac{1}{m}\Delta_{ij}^{(l)} & \text{if } j = 0 \end{cases}$

D is "accumulator". $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

Implementation details

↳ refer to notes and video

↳ No need to write code for hand-written notes.

↳ unrolling = you can make / convert between matrix / vector repn of matrices

↳ gradient checking: bug-free impl guarantee

↳ use random to set initial theta

Week 6

Evaluating a learning algorithm

ways to arrive at better hypothesis

↳ more examples

↳ more / less # of features

↳ more / less value of λ.

to evaluate a hypothesis, we split data into training set (70%) & test set (30%).

we ↳ learn θ, minimize $J_{train}(\theta)$ using training set

↳ Compute test set error $J_{test}(\theta)$

computing test set error

↳ lin. reg. : $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x)^{i}_{test} - y^{(i)}_{test})^2$

↳ log. reg :

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

$$\text{Test error} = \frac{1}{m_t} \sum_{i=1}^{m_{test}} err(h_\theta(x^{(i)}_{test}), y^{(i)}_{test})$$

Model Selection

you can break down data set into three data sets:

training set (60%), cross validation set (20%), test set (20%)

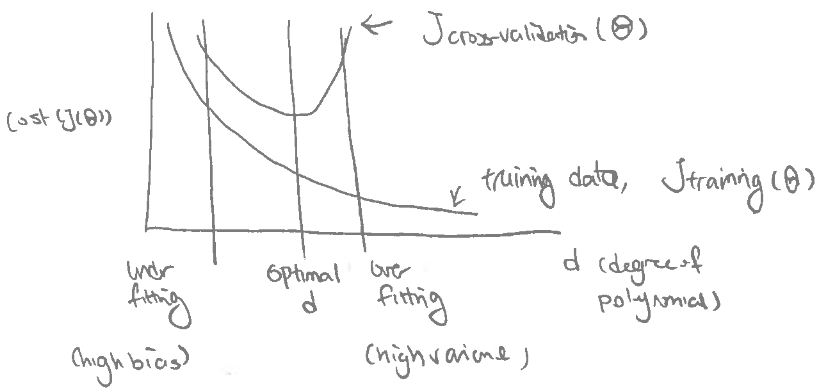
Idea: test different degree of polynomial, evaluate error function

1. optimize params in θ using training set for each degree
2. find the polynomial degree d that produce least error by cross validation set
3. estimate generalized error with $J_{test}(\theta^{(d)})$ using test set. ($d :=$ deg returning lowest error)

(this way, test set is NOT associated with the param training.

Diagnosing bias vs Variance.

If you have bad prediction, you need to figure out whether its high bias or variance.

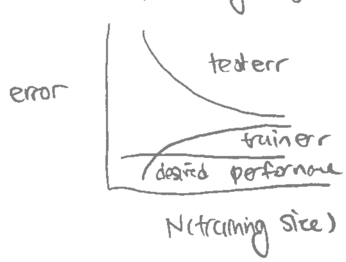


Regularization vs bias/Variance.

We use similar algorithm for testing regularization term λ.

learning curves

↳ experiencing high bias

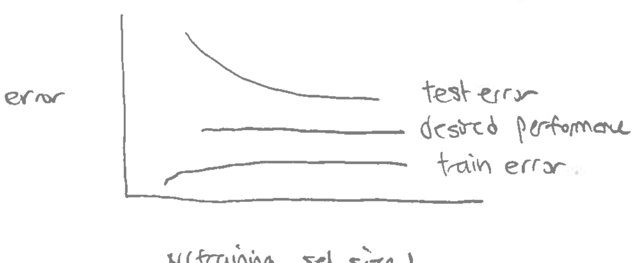


↳ low training set size causes $J_{train}(\Theta)$ low, $J_{cv}(\Theta)$ high.

↳ large training set size causes both $J_{train}(\Theta)$, $J_{cv}(\Theta)$ high but also $J_{train}(\Theta) \approx J_{cv}(\Theta)$

↳ getting more data won't help much.

↳ experiencing high variance



↳ low training set size: $J_{train}(\theta)$ low, $J_{cv}(\theta)$ high

↳ large training set size: $J_{train}(\theta)$ increase with set size, and $J_{cv}(\theta)$ continue to decrease without plateauing. $J_{train}(\theta) < J_{cv}(\Theta)$, but difference remains significant.

↳ getting more data will likely to help

Debugging learning algorithms.

problem	try
high var	get more training data
high var	less features
high bias	get more features
high bias	add poly features
high bias	decrease λ
high var	increase λ.

Small neural network: computationally cheap (prone to underfitting)
large neural network: computationally expensive (prone to overfitting, (use λ (regularization) to fix)

Building a spam classifier
↳ Designing ML system. (building your own system)
↳ identify features, (X) and classifyer (y)
↳ ways to spend more time
↳ collect lots of data
↳ more sophisticated features
↳ algorithms to process input data.

Error analysis
↳ implement a quick implementation
↳ use it to decide how to spend your time
↳ plot learning curves, and decide what to do.
↳ manually examine errors, analyse
↳ implement a metric that returns performance on different change/ideas.

Skewed classes
↳ case when one class has very large size, another very little size.
↳ different error metric → use true +, true − & false +, false − to classify (precision / recall) Precision: true + / pred +, * Recall true + / actual +

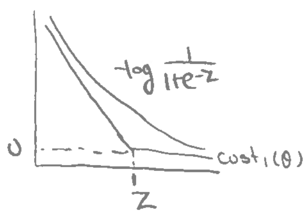
can change $h_\theta(X)$ threshold, which trade off precision / recall
precision metric: F, score: PR / (P+R)

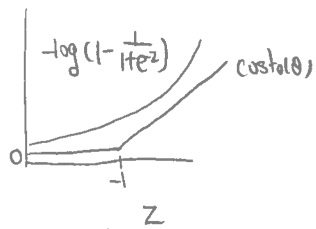
Week 7

Support vector machine (SVM)

↳ using cost(1), cost(0) similar to h_θ but easier computation wise

↳ ie. if $y=1$





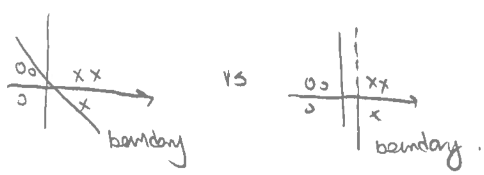
minimizing θ given by: optimization projective;

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

↳ large margin classifyer.

↳ allows a decision boundary that stays naturally far apart from dataset.

↳ the perpendicular vector is closest to examples.



Kernels

label landmark (defining feature) then use distance measure.

$f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$ if $\|w\| \approx$ small $f_i \approx 1$, large $f_i \approx 0$

learn non-linear decision boundary

predict 1 if close to L_1, L_2, L_3

0 otherwise.

details (SVM with kernels)

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

chose $l^1 = x^1, l_2 = x^2, \cdots l_m = x^m$

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

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

⋮

for $i=1, \cdots m$, $x^{(i)} = \begin{bmatrix} f_1^i \\ f_2^i \\ \vdots \\ f_m^i \end{bmatrix}$ $f = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{bmatrix}$ → $f_0 = 1$

<u>Hypothesis</u> given x, compute features $f \in \mathbb{R}^{m \times 1}$

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

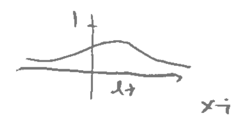
training using $f^{(i)}$'s similarity metric instead.

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

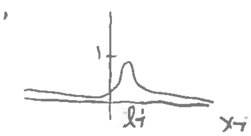
note that since $C = \frac{1}{\lambda}$, large $C \rightarrow$ low bias, high var (ie small λ)

small $C \rightarrow$ high bias, low var (ie large λ)

large $\sigma^2 \rightarrow$ features f_i vary smooth, high bias, low var



small $\sigma^2 \rightarrow$ features f_i vary less smooth, low bias, high var



your job is to choose C and σ^2

<u>using an SVM</u>

↳ nice software libraries: liblinear, libsvm.

↳ need to choose: kernel (use or no use) and parameter C

↳ linear kernel (no kernel)

↳ Gaussian kernel

↳ need to choose σ

↳ need to do feature scaling

↳ for other choices of kernel, it must satisfy Mercer's theorem so it for sure do not diverge

↳ multiclassification

↳ built in SVM package

↳ one-vs-all

↳ <u>logistic regression vs SVM.</u>

which to choose?

$\begin{cases} n = \# \text{ features} \\ m = \# \text{ training example.} \end{cases}$

SVM → convex function → return global optima

↳ if n large relative to m ($n \gg m$, $n \gtrsim 10{,}000$, $m \lesssim 10 \sim 1000$)

↳ use L.R, or S.V.M w/ linear kernel.

↳ if n small, m intermediate ($n = 1 \sim 1000$, $m = 10 \sim 1000$)

↳ use SVM w/ Gaussian kernel

↳ if n small m large ($n = 1 \sim 1000$, $m = 50{,}000+$)

↳ add more feature, then use LR or SVM with lin kernel

↳ nn works well with these settings, but is slow to train.

$\boxed{\text{Week 8}}$

unsupervised learning

↳ given input has no labels

↳ algorithm finds clustering data. / structure (as example of unsupervised learning algorithm

↳ used for identifying market segmentation, social network analysis, organize computer clusters.
galaxy formation / astronomical data analysis

K-means algorithm

↳ for clustering

↳ step 1: initial cluster centroids randomly

keep on iterating $\begin{cases} \text{step2: assign each point to centroid that's closer to it (assign } c^{i} \text{, to index of closest cluster centroid)} \\ \text{step3: move the centroids, to new mean of assigned points. mean} = \mu_K \end{cases}$

(step 2: 1~m loop; step 3: 1~K loop)

↳ input: K (# of clusters)
training set $\{x^1, \cdots x^m\}$
$x^i \in \mathbb{R}^n$

↳ also useful for non-separated clusters.
it can still separate out clusters, although may not seem like a obvious separation

Optimization objective

$c^{(i)}$ = index of cluster (1~K) that $x^{(i)}$ is currently assigned to

μ_K = cluster centroid K ($\mu_K \in \mathbb{R}^K$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster that $x^{(i)}$ is assigned to.

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

note that J does not increase ! as a func of iteration.

random initialize clustering centroid

steps: ↳ set K<m

↳ randomly choose K training examples

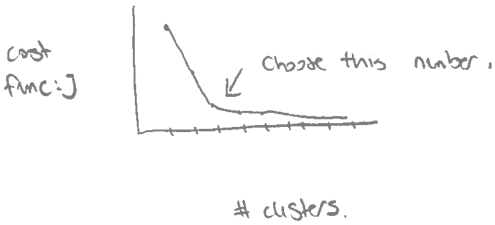
↳ set $\mu_1, \cdots \mu_K$ to these examples.

↳ K-means may not always end at global optima.

↳ avoid bad clustering, can initialize many times ! compute J each time & pick one that gives lowest cost.
when having small # of cluster, multiple / reinitialization helps the most.

Pick # of clusters

↳ Choosing # of clusters "elbow method"



↳ But often, the 'elbow' does not appear.

↳ don't expect it to work.

↳ better way: pick K from "what you want to do with result of the learning".

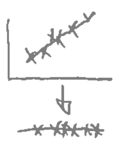
↳ i.e. how many groups you want? for what purpose are you running this algorithm

Data compression ← dimensionality reduction

↳ example: 2D-1D

↳ reducing redundant data

↳ represent datapoints in a line by the line (2D-1D). and project each point to the line



for each x_i, $x_i \in \mathbb{R}^2 \to z_i \in \mathbb{R}$

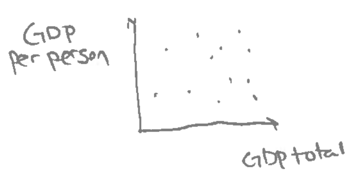
example: 3D space roughly on plane → lie on plane

you can generally reduce from high to low dimension given that data roughly lie on like-dimensions

Visualization ← dimensionality reduction.

↳ help with data visualization that looks very complex

↳ having pers in smaller dimension to capture info in more important feature in data



example: country's economy

↳ usually reduce to 2-D or 3-D so it's easy to visualize.

PCA (Principle component analysis)

↳ goal: find surface of lower dimension that has smallest sum of distance (from actual pt to projected pt)

↳ (projection error)

↳ note: PCA is NOT linear regression.

lin reg minimize vertical distance (predict y)

PCA minimize perpendicular distance (nothing to do with y)

PCA algorithm

↳ training set : $\{x^1, \cdots x^m\}$

↳ first, do preprocessing (feature scaling / mean normalization)

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

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

also scale the features

ie. $x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$

↳ then, we need to compute the vector u_i, and the projections / new representations.

Reduce data from dim n to dim k.

covariance matrix : $\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T$ ↙ nxn matrix

eigenvectors of Σ : [U,S,V] = svd(Sigma);

U will be a nxn matrix whose colums are $u^1, u_2, \cdots u^m$.

$$U = \begin{bmatrix} | & | & & | \\ \underbrace{u^1 \; u^2 \cdots}_{k} & & & u^m \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n\times n}$$

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

$$U_{reduce} = \underbrace{\begin{bmatrix} | & & | \\ u_1 & \cdots & u_k \\ | & & | \end{bmatrix}}_{k} \qquad z^{(i)} = U_{reduce}^T \, x^{(i)}$$

reconstruction from compressed representation

$Z = U_{reduce}^T * X$

$X_{approx} = U_{reduce} * Z$

Applying PCA

choosing K (# of principle components)

↳ average square projection error $= \frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2$

↳ total variance in data $= \frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2$

choose K to be smallest value, s.t.

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x^{(i)}_{approx}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01$$ "99% variance is retained"

algorithm

[U,S,V] = svd(sigma)   S is a diagnal matrix

Pick smallest K s.t.

$$\frac{\sum_{i=1}^{K} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$ (0.99 variance retained).

Use PCA to speed up learning algorithm

when running PCA, only run it on the training set.

to speed up:

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

extracted input: $x^1, x^2, \cdots x^m \in \mathbb{R}^{10000}$

$$\downarrow \text{PCA}$$

$$z^1, z^2, \cdots z^m \in \mathbb{R}^{1000}$$

then train $\{(z^1, y^1), \cdots (z^m, y^m)\}$ by $h_\theta(z)$

Summary of Application of PCA

↳ compression

↳ reduce storage to store data

↳ speed up learning algos.   } chose K by % variance

↳ visualization

↳ K=2 or K=3

↳ Note: DONT use PCA to prevent overfitting.

↳ it's a bad way of using PCA

↳ use regularization instead.

↳ note: when designing system

↳ first try without PCA

↳ only if it does not work, utilize PCA

↳ if you need to save storage space or time.