DS01 Machine Learning (stanford)

Week 1 Introduction and Univariate Linear Regression

- Algorithms:
- Supervised learning / Unsupervised learning / Reinforcement learning / Recommender system
- Supervised learning: "right answer" given
- Regression: predict continus valued output
- Classification: discrete valued output (0 or 1)
- Unsupervised learning:
- Kmean, cluster
- Linear regression w one variable
- Model representation:

Training Set
↓
Learning Algorithms
↓
input → h → output
(hypothesis)

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

* θ_0, θ_1 is parameter
* x is input / feature
* y or $h_\theta(x)$ is output

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

Goal: $\min_{(\theta_0, \theta_1)} J(\theta_0, \theta_1)$ in training set $h_\theta(x^i) = \theta_0 + \theta_1 x^i$

- Gradient descent algorithm

* repeat until converge: $\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \}$ (learning rate)
* simultaneously update θ_0 and θ_1

$$temp0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \theta_0 - \alpha \frac{1}{m} \sum_{i}^{m} (h_\theta(x^i) - y^i)$$

$$temp1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \theta_1 - \alpha \frac{1}{m} \sum_{i}^{m} (h_\theta(x^i) - y^i) x^i$$

$\theta_0 = temp0$

$\theta_1 = temp1$

update θ_0 and θ_1 simultaneously

* α too small ⇒ too slow to converge, α too large ⇒ fail converge

* Gradient descent can converge to local min, even with fixed α
  As we approach to local min, gradient descent will automatically take smaller step, so no need to decrease α over the time
* "Batch": each step of gradient descent use all training example

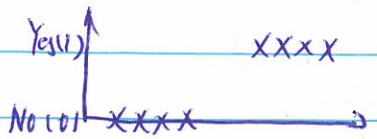
Week 2 Multivariate Linear Regression

- Linear regression with multiple variables
  - Hypothesis: $h_\theta(x) = \theta^T x$, parameter $\theta = \begin{bmatrix} \theta_0 \\ \theta_n \end{bmatrix}$, input $x = \begin{bmatrix} x_0 \\ x_n \end{bmatrix}$
  - cost function: $J(\theta) = \frac{1}{2m} \sum_i^m (h_\theta(x^i) - y^i)^2$
  - Gradient descent: Repeat $\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \}$
    Simultaneously update θ_j $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_i^m (h_\theta(x^i) - y^i) x_j^i$
- Feature Scaling: { make sure feature are on similar scale
  give every feature in $-1 \le x_i \le 1$
  - Mean normalization: $x_i = (x_i - \mu_i)/s_i$ where s_i = range (max-min) or std
- Learning rate
  - $J(\theta)$ should decrease after every iteration
    * Declare convergence if $J(\theta)$ decrease by $< 10^{-3}$ in 1 iteration
  - for smaller α, $J(\theta)$ should decrease but can be slow to converge
    for larger α, $J(\theta)$ may not decrease and may not converge
- Normal equation: solve θ analytically ← work well if small n.
  - $\frac{\partial}{\partial \theta_j} J(\theta) = 0$ solve $\theta_0 \cdots \theta_n$ where $J(\theta) = \frac{1}{2m} \sum_i^m (h_\theta(x^i) - y^i)^2$
  - $\theta = (X^T X)^{-1} X^T y$
  - $(X^T X)$ is non-invertible if { Redunt feature (linear dependent)
    too many features, delete some features or use regularization

Week 3 Logistic Regression

- Classification

- $y \in \{0, 1\}$ 0: Negative class (benign tumor)
  1: Positive class (malignant tumor)



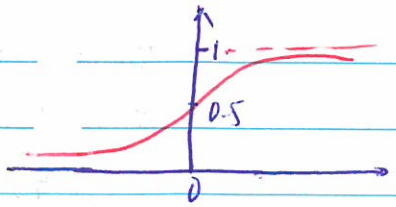
Threshold $h_\theta(x)$ at 0.5 $\begin{cases} h_\theta(x) \geq 0.5, & y=1 \\ h_\theta(x) < 0.5 & y=0 \end{cases}$

- Classification: $y=0$ or 1
  Logistic regression: $0 \leq h_\theta(x) \leq 1$

• Hypothesis Representation

- Logistic Regression Model

$$\left.\begin{array}{l} 0 \leq h_\theta(x) \leq 1 \\ h_\theta(x) = g(\theta^T x) \\ g(z) = \frac{1}{1+e^{-z}} \end{array}\right\} \Rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



* Sigmoid function / Logistic function

- Interpretation of Hypothesis output

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

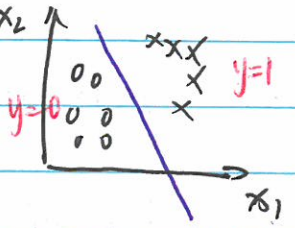
* $h_\theta(x) = P(y=1 \mid x; \theta)$ probability that $y=1$, given x, parameterized by θ

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

• Decision Boundary

- $h_\theta(x) = g(\theta^T x)$ $\Rightarrow$ $y=1$ if $h_\theta(x) \geq 0.5 \Rightarrow \theta^T x \geq 0$

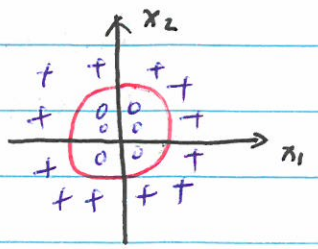
$g(z) = \frac{1}{1+e^{-z}}$ $y=0$ if $h_\theta(x) < 0.5 \Rightarrow \theta^T x < 0$

- 

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

predict $y=1$ if $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$

Decision Boundary



- Nonlinear Decision Boundary

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

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

• Cost Function

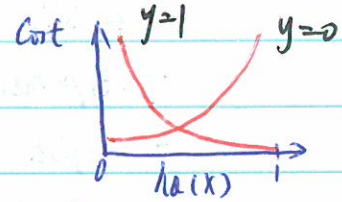
- Training set: $\{(x^1, y^1) \sim (x^m, y^m)\}$

$x \in \begin{bmatrix} x^1 \\ \vdots \\ x^m \end{bmatrix}$ $x_0 = 1$, $y \in \{0, 1\}$

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

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

* non-convex ↘ $\text{Cost}(h_\theta(x), y)$



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

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

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

• Simplified Cost Function and Gradient Descent

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

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

- $\text{cost}(h_\theta(x^i), y^i) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$ $y=0$ or 1

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

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

⇒
- Cost function $J(\theta) = -\frac{1}{m}\left[\sum_{i}^{m} y^i \log h_\theta(x^i) + (1-y^i)\log(1-h_\theta(x^i))\right]$
- To fit parameter θ: $\min_\theta J(\theta)$
- To predict new x: output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ $P(y=1|x;\theta)$

- Gradient Descent:

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

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

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

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

· Advanced Optimization

– Cost function $J(\theta)$, Want $\min_\theta J(\theta)$

Given θ, we compute $J(\theta)$, $\frac{\partial}{\partial \theta_j} J(\theta)$

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

– Optimization algorithms:

* Conjugate gradient / BFGS / L-BFGS

* Advantge: No need to pick α / much faster

* options = optimset('GradObj', 'on', 'MaxIter', '100')

· Multi-class classification: One-vs-all

– One-vs-all = one-vs-rest

$h_\theta^i(x) = P(y=i|x;\theta)$

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

· The Problem of Overfitting

– Reduce # of features: manually / Model selection algorithm

– Regularization: Keep all features, but reduce θ_j

· Regularization Cost Function

– Small values of $\theta_{1:n}$: simpler hypothesis / less prone to overfitting

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

* $\lambda \uparrow$, algorithm works fine / fails to eliminate overfitting / results in underfitting / Gradient descent fail to converge

· Regularized Linear Regression

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

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

– Normal Equation: $\theta = (X^T X + \lambda \underbrace{\begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots \end{bmatrix}}_{(n+1)\times(n+1)})^{-1} X^T y$

- Regularized Logistic Regression

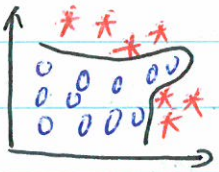
- $J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^i \log h_\theta(x^i) + (1-y^i)\log(1-h_\theta(x^i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

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

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

Week 4 Neural Networks I

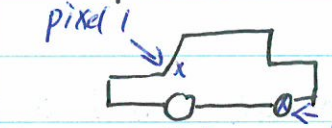
- Nonlinear Hypotheses:



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 x_2 + \dots + \theta_6 x_1 x_2^2 \dots)$

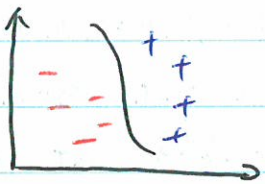
$O(n^2)$

- Computer Vision: Car detection (car vs not a car)



50 x 50 pixel image → 2500 pixels

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

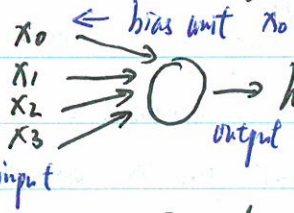


⇒ Quadratic features $(x_i x_j)$: 3 million features

- Neurons and the brain ⟸⟹ Neural Network

- Model Representation

- Neuron model: Logistic unit

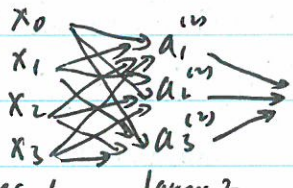


$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$ (weight)

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

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

- Neural Network



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

$\theta^{(j)}$ = matrix of weights from layer j to $j+1$

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

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

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

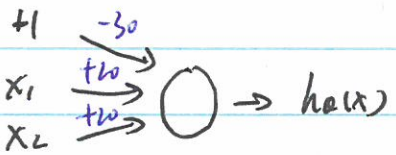
* if network has s_j unit in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

— Forward propagation : Vectorized implementation

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

· Example :

— AND : $x_1, x_2 \in \{0, 1\}$, $y = x_1$ AND x_2



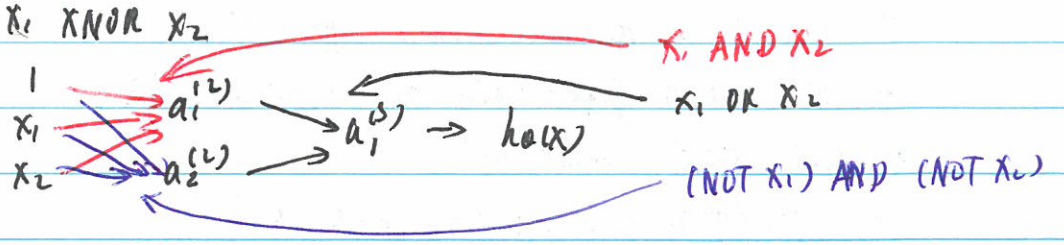
$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

x_1	x_2	$h_\Theta(x)$
0	0	≈0
0	1	≈0
1	0	≈0
1	1	≈1

— OR : $x_1, x_2 \in \{0, 1\}$. $y = x_1$ OR x_2

— NOT x_1

— x_1 XNOR x_2



· Multi-class classification

— One-vs-all : $h_\Theta(x) = \begin{bmatrix}1\\0\\0\end{bmatrix}$ $h_\Theta(x) = \begin{bmatrix}0\\1\\0\end{bmatrix}$ $h_\Theta(x) = \begin{bmatrix}0\\0\\1\end{bmatrix}$

when pedestrian when car when motorcycle

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

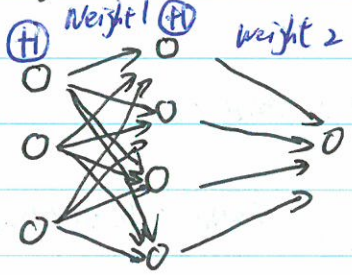
where $y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\end{bmatrix}$

pedestrian car motorcycle truck

· How to build your own Neural Network from scratch in Python

— What is a Neural Network

* Neural Network consists of
  - Input layer, x
  - Arbitrary amount of hidden layer
  - output layer $\hat{y}$
  - A set of weights and biases between each layer, W and b (similar to Θ in the course)
  - Activation function for each hidden layer σ (Sigmoid activation function)
* 2-layers neural network



Input Layer Hidden Layer Output layer

- Training the Neural Network

$$\hat{y} = \sigma(W_2\,\sigma(W_1 x + b_1) + b_2)$$

* W and b are the only variables affect $\hat{y}$
* We need to fine-tuning the weight and biases
* each iteration consists of
  - feedforward: calculating predicted $\hat{y}$
  - back propagation: updating weight and biases

$$\begin{matrix} x \\ W \\ b \end{matrix} \rightarrow W_1 x + b_1 \rightleftarrows z = \sigma(W_1 x + b_1) \rightleftarrows W_2 z + b_2 \rightleftarrows \hat{y} = \sigma(W_2 z + b_2) \rightleftarrows \text{loss}(y, \hat{y})$$

→ Feedforward ← Backpropagation

- Feed Forward

* $\hat{y} = \sigma(W_2\,\sigma(W_1 x + b_1) + b_2)$

- Loss function

* Sum-of-square error $= \sum_1^n (y - \hat{y})^2$

* or $-\frac{1}{m}\sum_{i}^{m}[y^i \log a_i^L + (1-y^i)\log(1-a_i^L)]$

* Goal: find best set of weights and biases to min Loss function

- Backpropagation

* Use Loss function to measure the error of prediction, and then propagate the error back and update weight and biases

o Derivative of loss function to weight and biases

o and use Gradient descent to update weight and biases

* Use Chain rule to get Derivative

$$Loss(y,\hat{y}) = \sum_{i}^{n}(y-\hat{y})^2$$

$$\frac{\partial Loss(y,\hat{y})}{\partial w} = \frac{\partial Loss(y,\hat{y})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w} \quad \text{where } z = wx+b$$

$$= 2(y-\hat{y}) * \text{derivative of sigmoid function}$$

* $O \longrightarrow O \xrightarrow{C_0}$ Output $\hat{y}$ $\qquad C_0 = (a^L - y)^2$

a^{L-1} $\quad a^L$ $\quad$ Desired Output y

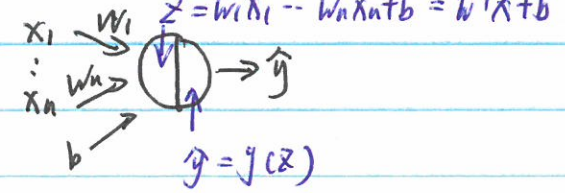
$$z^L = w^L a^{L-1} + b^L$$

$$\hat{y} = a^L = \sigma(z^L)$$

· Deep Neural Network from scratch in Python

- a_i^L L is Lth layer, i is ith example

- Single neuron



* A neuron calculate $z = w^Tx + b$ followed by an activation function

* g is activation function (Sigmoid, tanh, ReLU)

- Work flow:

1. normalized all feature (input): to center and standardize dataset $\frac{x-\bar{x}}{\sigma}$ define the inputs

2. Initialize parameters and define hyperparameters:
   * # of iteration
   * # of layers L in the neural network
   * size of hidden layer
   * Learning rate α
3. Loop for num_iteration:
   * Forward propagation (Feed forward) to calc current Loss
   * Compute cost function
   * Backward propagation (Back propagation) to calc current gradient
   * Update parameters
4. Use trained parameter to predict labels

- Activation functions

Sigmoid: * def sigmoid(z):
    S = 1/(1+np.exp(-z))
    return S

def sigmoid_backward(dA, z)
    S = sigmoid(z)
    dS = S*(1-S)
    return dS·dA

ReLU: * def relu(z):
    R = np.maximum(0,z)
    return R

def relu_backward(dA, z)
    dz = np.array(dA, copy=True)
    dz[z<=0] = 0
    return dz

- Forward propagation
  * $z^l = W^l A^{l-1} + b^l$
    $A^l = g(z^l)$
- Loss function
  * $-\frac{1}{m}\sum_i^m [y^i \log a_i^L + (1-y^i) \log (1-a_i^L)]$
- Backward propagation
  * using chain rule
- Update parameter: parameter_i = parameter_i - learning_rate * grads

- To avoid overfitting

* use regularization:

o L2 regularization: appropriately modify cost function

o drop out: randomly shuts down some neurons in each iteration

Week 5 Neural Network 2 and Practical Advice

• Neural Network Learning — Cost Function

- Initialization

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

L = total # of layers

s_l = # of units (not include bias unit) in layer l

* Binary classification: $y = 0$ or 1

* Multi-class classification (K class): $\begin{bmatrix}1\\0\\0\end{bmatrix}$ car $\begin{bmatrix}0\\1\\0\end{bmatrix}$ truck ---

- Cost function:

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

Neural network: $J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log(h_\Theta(x^i))_k + (1-y_k^i)\log(1-h_\Theta(x^i))_k\right]$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{l})^2$$

• Neural Networks Learning — Backpropagation Algorithm

- Goal: $J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log(h_\Theta(x^i))_k + (1-y_k^i)\log(1-h_\Theta(x^i))_k\right]$

$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{l})^2$$

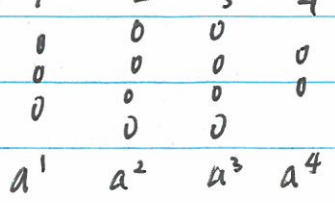
$\min J(\Theta)$

need to calculate: $J(\Theta)$ and $\frac{\partial}{\partial \Theta_{ij}^{l}} J(\Theta)$

- Forward propagation:

Given one training example (x, y)

$a^1 = x$



$z^2 = \Theta^1 a^1$

$a^2 = g(z^2)$ add a_0^2 (usually $=1$)

$z^3 = \Theta^2 a^2$

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

$z^4 = \Theta^3 a^3$

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

Layer	1	2	3	4
	o	o	o	
	o	o	o	o
	o	o	o	o
	o	o	o	
		δ^2	δ^3	δ^4
	a^1	a^2	a^3	a^4

- Back propagation Algorithm

  * δ_j^l = "error" of node j in layer l

Chain Rule ↓ $\delta_j^4 = a_j^4 - y_j = (h_\Theta(x))_j - y_j$

$\delta^3 = (\Theta^3)^T \delta^4 .* g'(z^3)$ ← if $g(\cdot)$ = sigmoid, $g'(z^3) = a^3 .* (1-a^3)$

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

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

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

Set $\Delta_{ij}^l = 0$ ← used to update $\frac{\partial}{\partial \Theta_{ij}^l} J(\Theta)$

for $i = 1$ to m:

set $a^1 = x^i$

perform forward propagation to compute a^l for $l = 2, 3 \cdots L$

using y^i, compute $\delta^L = a^L - y^i$

compute $\delta^{L-1}, \delta^{L-2}, \delta^{L-3} \cdots \delta^2$

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

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

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

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

  * δ_j^l = "error" of cost for a_j^l (unit j in layer l)

Formally, $\delta_j^l = \frac{\partial}{\partial z_j^l} \text{cost}(i)$, where $\text{cost}(i) = y^i \log h_\Theta(x^i) + (1-y^i) \log h_\Theta(x^i)$

- Implementation Note:

  * Advanced optimization

function jVal, gradient J = costFunction (theta)

optTheta = fminunc(@costFunction, initialTheta, options)

$\Theta^1, \Theta^2, \Theta^3$ - matrices (Theta1, Theta2, Theta3) $\vec{R}^{10\times11}, \vec{R}^{10\times11}, \vec{R}^{1\times11}$

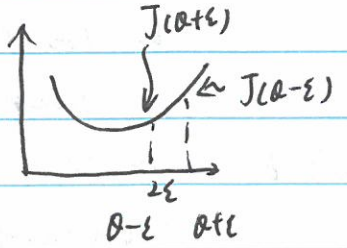
D^1, D^2, D^3 - matrices (D1, D2, D3) $\vec{R}^{10\times11}, \vec{R}^{10\times11}, \vec{R}^{1\times11}$

* From thetaVec, get $\Theta^1, \Theta^2, \Theta^3$
* Use forward / backpropagation to compute D^1, D^2, D^3 and $J(\Theta)$. Unroll D^1, D^2, D^3 to get gradientVec

- Gradient Check

* $\frac{d}{d\Theta} J(\Theta) = J(\Theta+\varepsilon) - J(\Theta-\varepsilon)/2\varepsilon$

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



* $\Theta = [\Theta_1 \dots \Theta_n]$

$\frac{d}{d\Theta_1} J(\Theta) \approx \frac{J(\Theta_1+\varepsilon, \Theta_2 \dots \Theta_n) - J(\Theta_1-\varepsilon, \Theta_2 \dots \Theta_n)}{2\varepsilon}$

$\frac{d}{d\Theta_n} J(\Theta) \approx \frac{J(\Theta_1 \dots \Theta_{n-1}, \Theta_n+\varepsilon) - J(\Theta_1 \dots \Theta_{n-1}, \Theta_n-\varepsilon)}{2\varepsilon}$

check that gradApprox $\approx$ DVec

• Neural Network Learning — Random Initialization

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

0 Theta1 = rand(10,11) * (2*INIT_EPSILON) - INIT_EPSILON

• Neural Network Learning - Putting it together

- # of input features and normalize all feature

# of output

# of hidden layers, have same # of hidden units in each layer (usually the more, the better)

- Work Flow

1. Randomly initialize weight
2. Implement forward propagation to get $h_\Theta(x^i)$ for any x^i
3. Implement to compute cost function $J(\Theta)$
4. Implement backprop to compute $\frac{\partial}{\partial \Theta^l_{ij}} J(\Theta)$

   for $i = 1:m$

   $\Delta^l := \Delta^l + \delta^{l+1}(a^l)^T$
5. Using gradient check to compare $\frac{\partial}{\partial \Theta} J(\Theta)$ from back propagation vs numerical estimate of gradient of $J(\Theta)$
6. Use gradient descent or advanced optimization method with back propagation to min $J(\Theta)$

Week 6

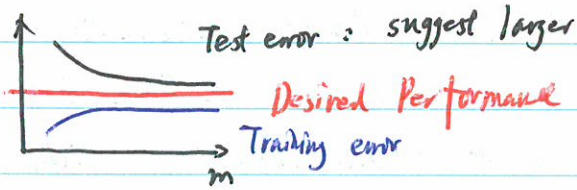
- Advice for Applying Machine Learning

— { Overfitting : high variance
  { Too few features to classify spam (high bias)

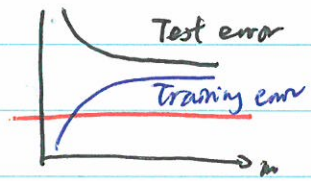
* Diagnostic:

{ Variance : training error will be much lower than test error
{ Bias : training error will be high.

* Learning curve for high variance



or



* { Fix high variance : more training example / smaller set of feature / ↑ λ
  { Fix high bias : larger set of feature / email header feature / ↓ λ
  { Fix optimization algorithm : more iteration on gradient descent / Newton's method / different λ / SVM

Procedure: — *Data set is split into training set (70%) and test set (30%)

* Learning parameter Θ from training set (min training error $J(\Theta)$)

* Compute test set error:

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

* Misclassification error (0/1 misclassification error)

- Model Selection and Training / Validation / Test set

* To avoid overfit problem

1. Split dataset:
   - Training set (60%): $(x^1, y^1) \cdots (x^m, y^m)$
   - Cross validation set (20%): $(x^1_{cv}, y^1_{cv}) \cdots (x^{m_{cv}}_{cv}, y^{m_{cv}}_{cv})$
   - Test set (20%): $(x^1_{test}, y^1_{test}) \cdots (x^{m_{test}}_{test}, y^{m_{test}}_{test})$

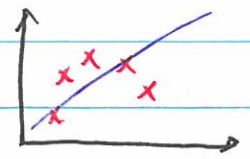
2.
   - Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i}^{m} (h_\theta(x^i) - y^i)^2$
   - Cross Validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i}^{m_{cv}} (h_\theta(x^i_{cv}) - y^i_{cv})^2$
   - Test error: $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i}^{m_{test}} (h_\theta(x^i_{test}) - y^i_{test})^2$

3. pick the model: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_5 x^5$

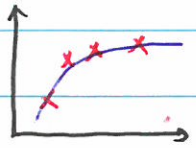
4. Report test set error: $J_{test}(\theta^5)$ and to be an optimistic estimate of generalization error.

- Diagnostic bias vs Variance

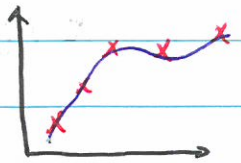
*



High Bias (underfit)

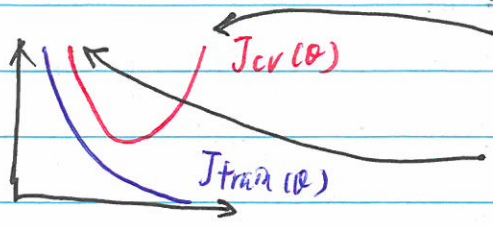


Just right



High Variance (overfit)

*



Variance: $J_{train}(\theta)$ is small

$J_{cv}(\theta) \gg J_{train}(\theta)$

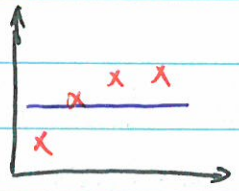
Bias: $J_{train}(\theta)$ is high

$J_{cv}(\theta) \approx J_{train}(\theta)$

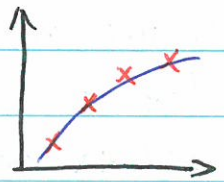
- Regularization and Bias / Variance

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

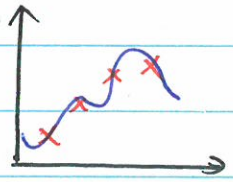
$$J(\theta) = \frac{1}{2m} \sum_{i}^{m} (h_\theta(x^i) - y^i)^2 + \lambda / 2m \sum_{j}^{n} \theta_j^2$$



large λ → high bias
underfit



Intermediate λ
"Just right"



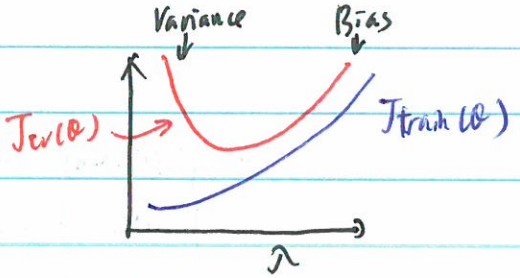
Small λ → high variance
overfit

* try different λ → $\min_{\theta} J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$ and choose with one
→ Test error: $J_{test}(\theta^{(1)})$

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

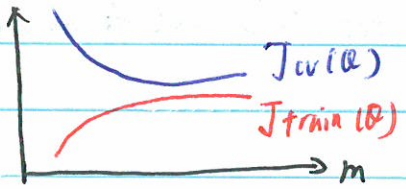
$J_{train}(\theta) = \frac{1}{2m}\sum_{i}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

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



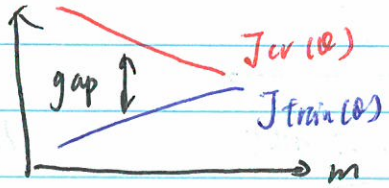
- Learning curve

* High bias



If suffering high bias, ↑m doesn't help

* High variance



If suffering high variance, ↑m is likely to help

- small neural network → computationally cheap → underfit
large neural network → computationally expense → overfit

Week 6 • Machine Learning System Design

- Spam classifier

* Supervised Learning. x = features of email, y = spam (1) or not spam (0)

Features x: choose 100 words indicative of spam / not spam

$x_j = \begin{cases} 1 & \text{if word } j \text{ appears} \\ 0 & \text{otherwise} \end{cases}$

- Error Analysis

* Recommended approach:

1. Implement the algorithm and test it on cross-validation data
2. Plot learning curve to decide if more data, more features to help
3. Error analysis: manually examine the examples in cross validation set. See if you spot any systematic trend in what type of error

* Using "stemming" software: universe / university

- Error Metrics for Skewed Classes

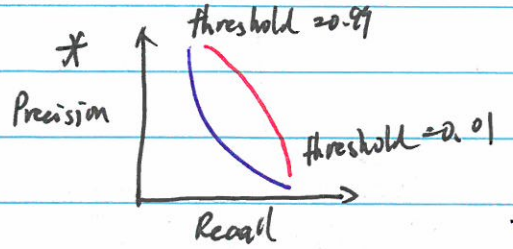
* Precision / Recall

predicted class	Actual class: 1	Actual class: 0
1	True Positive	False Positive
0	False Negative	True negative

Precision = (True Positive) / (Predicted Positive) = (True Positive) / (True Positive + False Positive)

Recall = (True Positive) / (Actual Positive)

= (True Positive) / (True Positive + False Negative)



Predict 1 if $h_\theta(x) \geq$ threshold

* To predict $y=1$ only if very confident ⇒ higher precision, lower recall

To avoid missing too many cases of $y=1$ (avoid false negative) ⇒ higher recall, lower precision

* F_1 Score (F Score) $= 2 \cdot P \cdot R / (P+R)$

- Large data rationale

* Using many parameters for LR or many hidden layers for NN, ⇒

$\Rightarrow$ Lower bias and $J_{train}(\theta)$ is small

* using large training set $\Rightarrow$ low variance $\Rightarrow$ $J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ is small

Week 7 Support Vector Machine

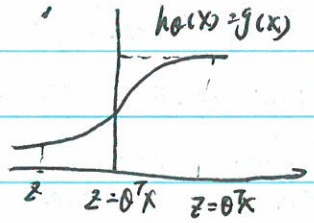
· Optimization objective

– Alternative view of logistic regression

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

If $y=1$. $h_\theta(x) \approx 1$. $\theta^T x \to \infty$

If $y=0$. $h_\theta(x) \approx 0$, $\theta^T x \to -\infty$



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

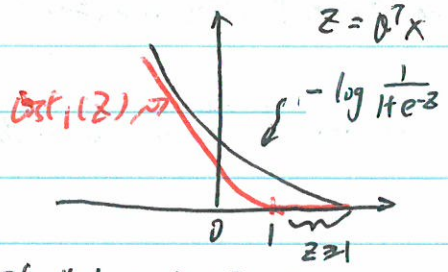
– Logistic regression:

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

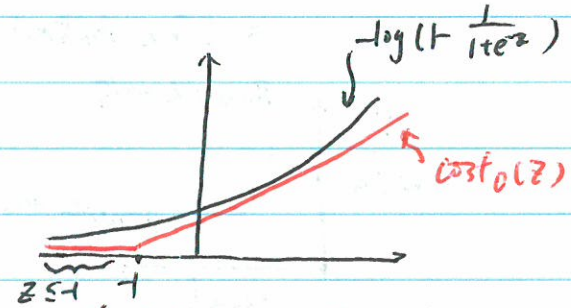
Support vector machine

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

· Large Margin Intuition



If $y=1$, want $\theta^T x \geq 1$



If $y=0$, want $\theta^T x \leq -1$

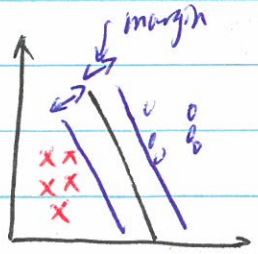
– Whenever $y^i=1$ and $\theta^T x^i \geq 1$ or $y^i=0$ and $\theta^T x^i \leq -1$

$$\text{cost} = \min_\theta 0 + \frac{1}{2}\sum_i^n \theta_j^2$$

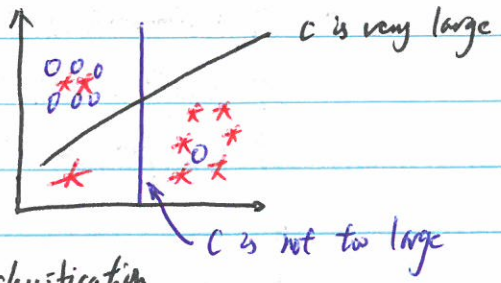
s.t. $\theta^T x^i \geq 1$ if $y^i=1$

$\theta^T x^i \leq -1$ if $y^i=0$

– Decision boundary:



2f with outlier



- The mathematics behind large margin classification
  - $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, p is the length of projection of v onto u

    $u^T v = u_1 v_1 + u_2 v_2 = p \cdot \|u\|$ where $\|u\| = \sqrt{u_1^2 + u_2^2}$
  - SVM Decision boundary: $\min_\theta \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 = \frac{1}{2}(\theta_1^2 + \theta_2^2) = \frac{1}{2}\|\theta\|^2$

    s.t. $\theta^T x^i \geq 1$ if $y^i = 1$

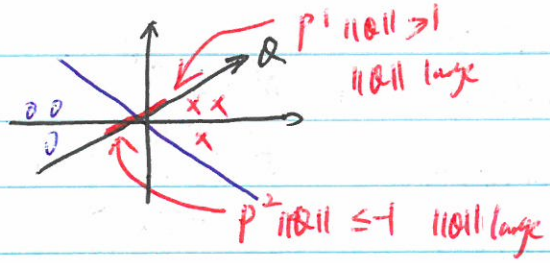
    $\theta^T x^i \leq -1$ if $y^i = 0$

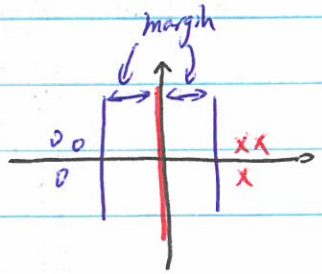
    where $\theta^T x^i = \theta_1 x_1 + \theta_2 x_2 = p^i \|\theta\|$

⇒ s.t. $p^i \|\theta\| \geq 1$ if $y^i = 1$

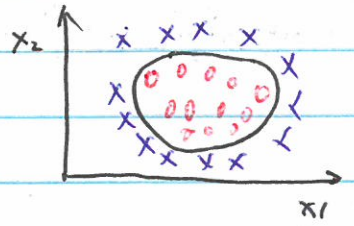
$p^i \|\theta\| \leq -1$ if $y^i = 0$

where p^i is the projection of x^i on to θ





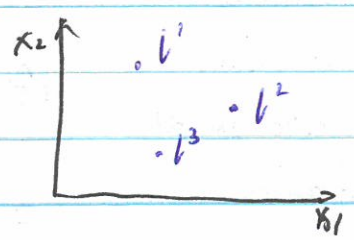
- Kernels



$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \theta_4 f_4 + \theta_5 f_5$

$f_1 = x_1$. $f_2 = x^2$ $f_3 = x_1 x_2$. $f_4 = x_1^2$. $f_5 = x_2^2$



Given x, compute new features depending on proximity to landmark, l^1, l^2, l^3

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

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

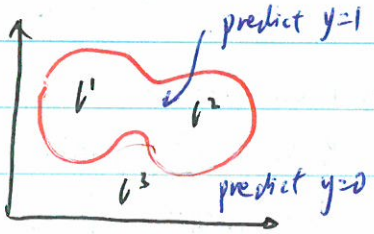
$f_3 = \text{similarity}(x, l^3) = \exp\left(-\frac{\|x - l^3\|^2}{2\sigma^2}\right)$

Gaussian kernel

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

  if $x \approx l^1$: $f_1 \approx 1$

  if x is far from l^1: $f_1 \approx 0$



Predict "1" when

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

- To choose l^1, l^2, l^3

* SVM with kernels

Given (x^1, x^1) -- (x^n, y^n) ⇒ choose $l^1 = x^1$ -- $l^m = x^m$

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

$\vdots$

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

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

For training example (x^i, y^i):

$x^i \rightarrow$

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

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

$\vdots$

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

$\vdots$

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

$\Rightarrow x^i = 2R^{n+1}$

$f^i = \begin{bmatrix} f_1^i \\ \vdots \\ f_m^i \end{bmatrix} 2R^n$ and $f_0^i = 1$

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

Training $\min_\theta \left(\sum_i^m y^i \text{cost}_1(\theta^T f^i) + (1-y) \text{cost}_0(\theta^T f^i) \right) + \frac{1}{2} \sum_j^m \theta_j^2$

where $\sum_j^m \theta_j^2 = \theta^T \theta = \|\theta\|^2$

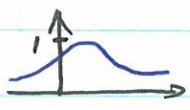
* SVM Parameters:

$C (= 1/\lambda)$: Large C: lower bias, high variance

Small C: high bias, lower variance

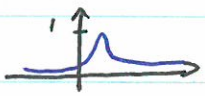
σ^2: large σ^2: features f_i vary more smoothly

High bias, lower variance



Small σ^2: feature f_i vary less smoothly

lower bias, higher variance



• Using SVM package (liblinear, libsvm) to solve for parameter θ

- specify parameter C and choice of kernel

* Gaussian Kernel: $f_i = \exp\left(-\frac{\|x - l^i\|^2}{2\sigma^2}\right)$ where $l^i = x^i$

need choose σ^2

* Note: do perform feature scaling before Gaussian kernel

- Other choice of kernel (need to satisfy "Mercer's Theorem")

* Polynomial kernel $k(x, l) = (x^T l)^2$ or $(x^T l + \text{constant})^{\text{degree}}$

* String kernel, chi-square kernel, histogram intersection kernel

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

or use one-vs-all method

- n = # of features, m = # of

* if $n >> m$, use logistic regression or SVM w/o kernel (Predict $y=1$ if $\theta^T x \geq 0$)

if n is small, m is intermediate: use SVM w Gaussian kernel

if $n << m$, add more features. use logistic regression or SVM w/o kernel

* Neural network work well for most, but slower to train

Week 8 Unsupervised Learning: Kmean cluster and PCA

• Unsupervised Learning: Clustering

- K-means Algorithm

* Input: { K (# of clusters)
training set $\{x^1, x^2 \cdots x^m\}$

* Randomly initialize K clusters $\mu_1, \mu_2 \cdots \mu_K$

Repeat {

Cluster assignment step: for $i = 1$ to m

c^i = index of cluster centroid closest to x^i ($\min_k \|x^i - \mu_k\|^2$)

Move centroid: for $k = 1$ to K

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

}

- Optimization objective

* c^i = index of cluster to which examples x^i is currently assigned

  μ_k = cluster centroid k

  μ_{c^i} = cluster centroid of cluster to which example x^i is assigned

* Optimization objective: $x^i \Rightarrow 5 \Rightarrow c^i = 5 \Rightarrow \mu_{c^i} = \mu_5$

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

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

* Cluster assignment step: min J cost $c^1, \dots, c^m$ (holding $\mu_1, \dots, \mu_K$ fixed)

  Move centroid: min J cost $\mu_1, \dots, \mu_K$

- Random Initialization:

* should have $K < m$

* Randomly pick K training example

  Set $\mu_1, \dots, \mu_K$ equal to these K examples

* For $i = 1$ to 100 {

  Randomly initialize k-means

  Run k-means, Get $c^1, \dots, c^m, \mu_1, \dots, \mu_K$

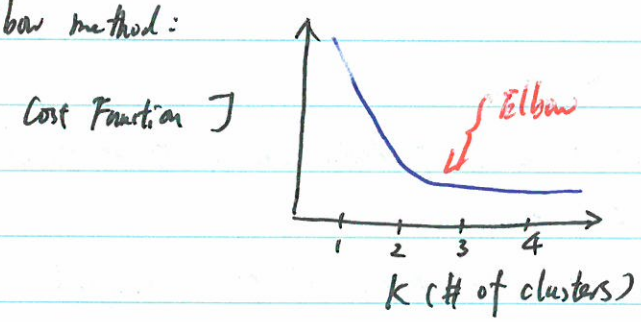
  Compute cost function (distortion) $J(c^1, \dots, c^m, \mu_1, \dots, \mu_K)$

  }

Pick clustering that gave lowest cost $J(c^1, \dots, c^m, \mu_1, \dots, \mu_K)$

- Choosing the # of clusters

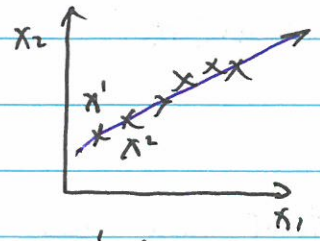
* Elbow method:



· Dimensionality Reduction

- Motivation

* Data Compression



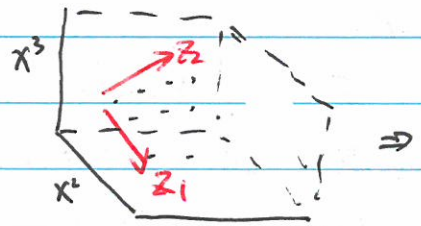
Reduce data from 2D to 1D

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

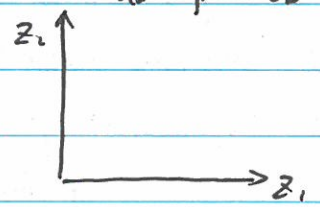
$x^2 \in \mathbb{R}^2 \rightarrow z^2 \in \mathbb{R}$

$\vdots$ $\vdots$

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



Reduce data from 3D to 2D



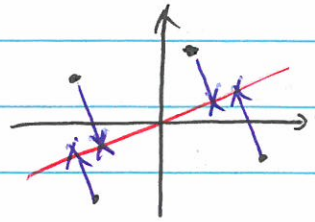
$x^i \in \mathbb{R}^3$ $\quad$ $z^i \in \mathbb{R}^2$

* Data Visualization

Country vs GDP (x_1), per capita GDP (x_2), Life (x_3), Poverty (x_4), Mean House Income (x_5)

Reduce data from 5D to 2D (z_1, z_2), easy to plot

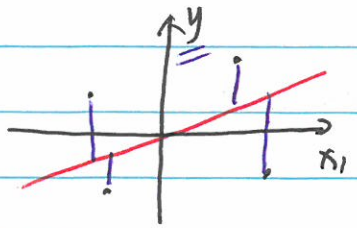
- Principal Component Analysis Problem Formulation

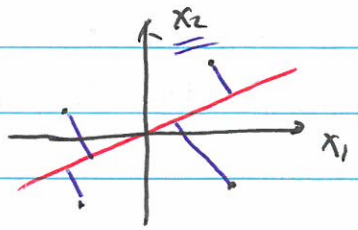


* Reduce from 2D to 1D: find a direction (u^1) onto which to project the data so as to min projection error.

* Reduce from nD to kD: find k vectors ($u^1 - u^k$) onto which to project the data so as to min projection error

* PCA is not linear regression





- Principal Component Analysis Algorithm

* Data Processing :

1. Train set : $x^1, x^2 \cdots x^m$
2. Preprocessing (feature scaling / mean normalization)

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

Replace x_j^i with $x_j^i - \mu_j$

If different features on different scales, scale features to have comparable range of values $x_j^i \leftarrow (x_j^i - \mu_j)/s_j$

* Reduce data from n-D to k-D.

3. Compute "covariance matrix" : $\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^i)(x^i)^T$ $\Rightarrow n \times n$ (x^i: $n \times 1$, $(x^i)^T$: $1 \times n$)
4. Compute "eigenvector" of matrix Σ : $[U, S, V] = svd(\Sigma)$ $U \in \mathbb{R}^{n \times n}$

If singular value decomposition : $eig(\Sigma)$

We get $U = [u^1, u^2 \cdots u^n] \in \mathbb{R}^{n \times n}$ $\rightarrow$ $U_{reduce} = [u^1, u^2 \cdots u^k]$

5. $x \in \mathbb{R}^n \Rightarrow z \in \mathbb{R}^k$ (k)

$$z^i = \underbrace{\begin{bmatrix} u^1 & u^2 & \cdots & u^k \end{bmatrix}^T}_{n \times k \text{ } U_{reduce}} x^i = \underbrace{\begin{bmatrix} -u^1- \\ -u^2- \\ \vdots \\ -u^k- \end{bmatrix}}_{k \times n} \underbrace{x^i}_{n \times 1}$$

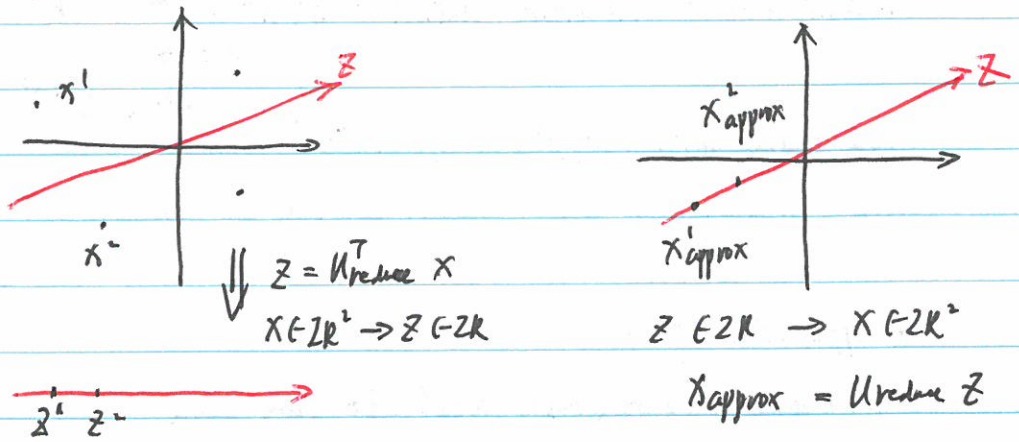
$k \times 1$

* 1. After mean normalization (ensure every feature has 0 mean) and optionally feature scaling

Σ (Sigma) $= \frac{1}{m}\sum_{i=1}^{m}(x^i)(x^i)^T = \frac{1}{m} * X' * X$

2. $[U, S, V] = svd(sigma)$
3. $U_{reduce} = U(:, 1:k)$
4. $z = U_{reduce}' * X$

- Reconstruction from Compressed Representation



- Choosing the # of Principal Components (k)

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

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

    Choose k to be the smallest value so that

    $$\frac{\frac{1}{m}\sum_{i}^{m} \|x^i - x_{approx}^i\|^2}{\frac{1}{m}\sum_{i}^{m} \|x^i\|^2} \overset{(*)}{\leq} 0.01 \quad (1\%) \Leftarrow 99\% \text{ of variance is retained}$$

  * Algorithm:

    1. try PCA with $k=1$
    2. Compute $U_{reduce}, z^1, z^2 \cdots z^m, x_{approx}^1 \cdots x_{approx}^m$
    3. check if $(*) \leq 0.01$

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

    $$S = \begin{bmatrix} S_{11} & & 0 \\ & \ddots & \\ 0 & & S_{nn} \end{bmatrix} \quad \text{(similar to eigenvalue)}$$

    For given k, $1 - \sum_{i}^{k} S_{ii} / \sum_{i}^{n} S_{ii} \leq 0.01$ or $\sum_{i}^{k} S_{ii} / \sum_{i}^{n} S_{ii} \geq 0.99$

    Pick smallest value of k for $\sum_{i}^{k} S_{ii} / \sum_{i}^{n} S_{ii} \geq 0.99$

- Advice for applying PCA

  * Speed up Supervised Learning

    Original train set: $(x^1, y^1) \cdots (x^m, y^m)$

Extract input: Unbalanced dataset: $x^1, x^2 \dots x^m \in \mathbb{R}^{10000}$

↓ PCA

$z^1, z^2 \dots z^m \in \mathbb{R}^{100}$

New training set: $(z^1, y^1) \dots (z^m, y^m)$

Note: Mapping $x^i \to z^i$ should be only on training set. This mapping can be applied to x^i_{cv} and x^i_{test} in cross validation and test sets.

* Application of PCA
  1. Reduce memory / disk need to store data
  2. Speed up learning algorithm
  3. Visualization

* Use z^i instead of x^i to reduce # of features to $k < n$, so to address overfitting. Bad ideas

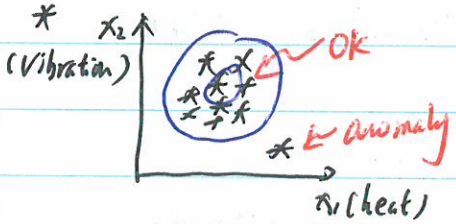
  use regularization instead $\min_\theta \frac{1}{2m}\sum_1^m (h_\theta(x^i) - y^i)^2 + \frac{\lambda}{2m}\sum_1^n \theta_j^2$

* Run Original data x^i first. if doesn't work. use PCA

Week 9 Anomaly Detection and Recommender System

• Anomaly detection

— Motivation



Aircraft engine feature: $x_1, x_2 \dots$

Dataset: $\{x^1, x^2 \dots x^m\}$

New engine: x_{test}

Model: $p(x)$

$p(x_{test}) < \varepsilon \Rightarrow$ Anomaly; $p(x_{test}) \geq \varepsilon \Rightarrow$ ok

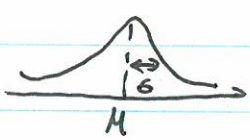
* Application: fraud detection / Manufacture / Monitoring computer in data center

— Gaussian distribution:

* $x \sim N(\mu, \sigma^2)$

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

$$\begin{cases} \mu = \frac{1}{m}\sum_1^m x^i \\ \sigma^2 = \frac{1}{m}\sum_1^m (x^i - \mu)^2 \quad \text{or } \frac{1}{m-1} \end{cases}$$



$\sigma \downarrow \Rightarrow$ sharp.

- Algorithm

* Density estimation

Training set: $\{x^1, \dots x^m\}$

$x_1 \sim N(\mu_1, \sigma_1^2)$ $x_2 \sim N(\mu_2, \sigma_2^2)$ $x_3 \sim N(\mu_3, \sigma_3^2)$

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

$= \prod_{1}^{n} P(x_j, \mu_j, \sigma_j^2)$

* Algorithm

1. Choose features x_i that might be indicative of anomalous examples
2. Fit parameter $\mu_1 \dots \mu_n$, $\sigma_1^2 \dots \sigma_n^2$

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

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

$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{1}^{m} x^i$

3. Given new example x, compute $P(x)$:

$P(x) = \prod_{1}^{n} P(x_j, \mu_j, \sigma_j^2)$

Anomaly if $P(x) < \varepsilon$

- Anomaly Detection system

* 1. Choosing feature x_i

2. Label anomalous vs non-anomalous (1 vs 0)
3. Traning set: $x^1 \dots x^m$ 60% good, 0% bad

Cross validation set: $(x_{cv}^1, y_{cv}^1) \dots (x_{cv}^{m_{cv}}, y_{cv}^{m_{cv}})$ 20% good, 50% bad

Test set: $(x_{test}^1, y_{test}^1) \dots (x_{test}^{m_{test}}, y_{test}^{m_{test}})$ 20% good, 50% bad

4. Fit model $P(x)$ on training set $\{x^1 \dots x^m\}$
5. On cross validation/test, predict

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

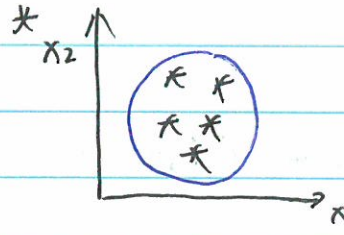
Possible metrics: { True positive, false positive, false negative, true negative
Precision / Recall
F_1-score

Note: Can use cross-validation set to choose ε

- Anomaly detection vs supervised Learning

* small # of positive ($y=1$), large # of negative ($y=0$)	Large # of positive and negative
* Many different type of anomalies	Algorithm can get a sense of
* future anomalities may look nothing like any previous ones	what positive examples are like, future is similar to old ones
* Fraud detection / Manufature / Monitor machine	Email spam / cancer / Weather prediction

- Choosing What Features to Use
  - * Non-gaussian features: $\log(x+c)$ / $\log$
  - * Error analysis:
    - $p(x)$ larger for normal, $p(x)$ small for anomalous
- Multivariate Gaussian Distribution

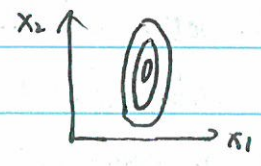


Don't model $p(x_1), p(x_2)$ separately, Model $p(x)$ all in one

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

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

* $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 0.6 & \\ & 1 \end{bmatrix}$



$\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$

* 1. Parameter: μ, Σ

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

2. Parameter fitting:

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

3 Given a x_{test} to $(*)$, flag anomaly if $p(x) < \varepsilon$

* For $\Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_n^2 \end{bmatrix}$

* Original Model	vs	Multivariate Gaussian
$\prod_{j=1}^{n} p(x_j, \mu_j, \sigma_j^2)$		$p(x, \mu, \Sigma)$
Independent feature		Capture correlation between features
Computationally cheap		Computationally expensive
OK even if m small		Must have $m > n$ or else Σ is non-invertible ($m \geq 10n$)

· Recommender Systems

- Problem formulation

* Predict movie ratings from 0 to 5 stars

n_u = no. users $\quad n_m$ = no. movies

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

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

- Content-based Recommendation

* For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^{n+1}$.
Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars

* Problem formular

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

$y(i,j)$ = rating by user j on movie i

$\theta^{(j)}$ = parameter vector for user j ($\theta^{(j)} \in \mathbb{R}^{n+1}$)

$x^{(i)}$ = feature vector for movie i

For user j, movie i, predicted rating $(\theta^{(j)})^T x^{(i)}$

m_j = # of movies rated by user j

Optimization objective: to learn $\theta^{(j)}$

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

to learn $\theta^1, \theta^2 \cdots \theta^{n_u}$:

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

Gradient descent update : $J(\theta^1, \cdots \theta^{n_u})$

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

$$\theta_k^j = \theta_k^j - \alpha \left\{ \sum_{i:r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right] + \lambda \theta_k^j \right\} \quad (\text{for } k \neq 0)$$

- Collaborative Filtering

* Given $\theta^1 \cdots \theta^{n_u}$, to learn x^i :

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

Given $\theta^1 \cdots \theta^{n_u}$ to learn $x^1 \cdots x^{n_m}$

$$\min \frac{1}{2} \sum_{i} \sum_{j:r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right]^2 + \lambda/2 \sum_{i} \sum_{k} (x_k^i)^2 \quad (1)$$

* Given $x^1 \cdots x^{n_m}$ can estimate $\theta^1 \cdots \theta^{n_u}$

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

Given $\theta^1 \cdots \theta^{n_u}$, can estimate $x^1 \cdots x^{n_m}$ (2)

$$\min \frac{1}{2} \sum_{i} \sum_{j:r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right]^2 + \lambda/2 \sum_{i} \sum_{k} (x_k^i)^2$$

Min $x^1 \cdots x^{n_m}$ and $\theta^1 \cdots \theta^{n_u}$ simultaneously :

$$J = (1) + (2)$$

* Algorithm :

1. Initialize $x^1 \cdots x^{n_m}$, $\theta^1 \cdots \theta^{n_u}$ to small random values
2. Min J using gradient descent.

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

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

3. For a user with θ and a movie with x, predict star rating of $\theta^T x$

- Vectorization : Low Rank Matrix Factorization

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

$$Y = (\theta^j)^T \cdot X^i = \begin{bmatrix} (\theta^1)^T x^1 & \cdots & (\theta^{n_m})^T \cdot x^1 \\ (\theta^1)^T x^2 & \cdots & (\theta^{n_m})^T \cdot x^2 \\ \vdots & & \vdots \\ (\theta^1)^T x^{n_m} & \cdots & (\theta^{n_m})^T \cdot x^{n_m} \end{bmatrix}$$

* How to find movie j related to movie i

  Smallest $\|x^i - x^j\| \Rightarrow$ movie i and j are similar

- Implementational Detail: Mean Normalization

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

Week 10 Large-scale Machine Learning

• Learning with large datasets

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i}^{m} (h_\theta(x^i) - y^i) x_j^i \quad \text{where } m = 100000000$$

• Stochastic Gradient Descent

- Linear regression with gradient descent

$$h_\theta(x) = \sum_{0}^{n} \theta_j x_j \qquad J(\theta) = \frac{1}{2m} \sum_{i}^{m} (h_\theta(x^i) - y^i)^2$$

Repeat {

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

} Batch Gradient Descent

call m examples

- Stochastic Gradient Descent

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

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

* 1. Randomly shuffle dataset

2. Repeat { for $i = 1 \cdots m$

$$\theta_j = \theta_j - \alpha [h_\theta(x^i) - y^i] \cdot x_j^i$$

use 1 example

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

• Mini-batch Gradient Descent

- Batch gradient descent: use all m examples in each iteration

- Stochastic gradient descent: use 1 examples in each iteration

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

* Let $b=10$ examples $(x^i, y^i) \sim (x^{i+9}, y^{i+9})$

$$\theta_j = \theta_j - \alpha \frac{1}{10} \sum_{i}^{i+9} (h_\theta(x^k) - y^k) x_j^k$$

or $i = 1, 11, 21, 31, \ldots, 991$

• Stochastic Gradient Descent Convergence

- Batch gradient descent : plot $J(\theta)$ vs # of iteration
- Stochastic gradient descent :

* During learning, compute $cost(\theta, (x^i, y^i))$ before updating θ using (x^i, y^i)

* Every 1000 iteration, plot $cost(\theta, (x^i, y^i))$ averaged over last 1000 example

* Learning rate α is held constant. Can slowly decrease α over time if we want θ to converge ($\alpha = const1 / (iteration\ number + const2)$)

Note. as iteration number ↑, $\alpha \to 0$

• Online Learning

- Shipping service : feature x. use service ($y=1$). not ($y=0$)

Learn $p(y=1 | x; \theta)$ to optimize price

- Product search : feature x, click link ($y=1$) not ($y=0$)

Learn $p(y=1 | x; \theta)$

• Map reduce and Data Parallelism

Batch gradient descent : $\theta_j = \theta_j - \alpha \frac{1}{400} \sum_{1}^{400} (h_\theta(x^i) - y^i) x_j^i$

Machine 1 : $temp_j^1 = \sum_{1}^{100} (h_\theta(x^i) - y^i) x_j^i$

Machine 2 : $temp_j^2 = \sum_{101}^{200} (h_\theta(x^i) - y^i) x_j^i$

Machine 3 : $temp_j^3 = \sum_{201}^{300} (h_\theta(x^i) - y^i) x_j^i$

Machine 4 : $temp_j^4 = \sum_{301}^{400} (h_\theta(x^i) - y^i) x_j^i$

$\theta_j = \theta_j - \alpha/400 \sum_{1}^{4} temp_j^i$

Multi-core machine : Core 1, Core 2, Core 3, Core 4