1 Have $J(\theta_0, \theta_1)$. and want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$.

Gradient Descent

Approach: 1 Choose some initial condition.

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

Repeat until convergence {

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

} where α is the learning rate.

Linear Regression

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

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

So

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

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

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

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

And now can use these formulas in gradient descent.

If you use all of your training examples, this is batch gradient descent.

Q. $(x, y) \in ((3,2),(1,2),(0,1),(4,3))$

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

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

What is $J(0,1)$?

A $J(\theta_0,\theta_1) = \frac{1}{8}((3\theta_1 + \theta_0 - 2)^2 + (\theta_1 + \theta_0 - 2)^2 + (\theta_0 - 1)^2 + (4\theta_1 + \theta_0 - 3)^2)$

sub in θ_0, θ_1 $= \frac{1}{8}(1^2 + (-1)^2 + (-1)^2 + 1^2)$

$= \frac{1}{2}$

$h_\theta(4) = -1 + 0.5 \times 4$ (where $\theta_0 = -1$ & $\theta_1 = 0.5$)

$= 1$

Notation

Now let's assume we have multiple features, e.g. size, age, number of floors...

n = number of features

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

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

Hypothesis (Multivariate Linear Regression)

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

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

$= \theta_0 + \sum_{j=1}^{n} \theta_j x_j$

or

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

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

Cost Function

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

Gradient Descent

Repeat {

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

}

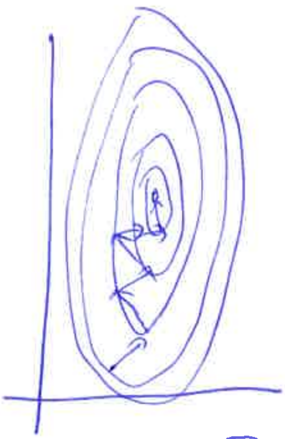
so $\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

(Simultaneously update θ_j for $j = 0, \dots, n$

Coding efficiencies

If features are on a similar scale, gradient boosting converges more quickly.

i.e.



or

So we can use Feature Scaling! Get every feature into approximately $-1 \leq x_i \leq 1$. e.g. $x_0 = 1$

$0 \leq x_1 \leq 3$ ✓

$-2 \leq x_2 \leq 0.5$ ✓

$-100 \leq x_3 \leq 100$ ✗

$-0.0001 \leq x_4 \leq 0.0001$ ✗

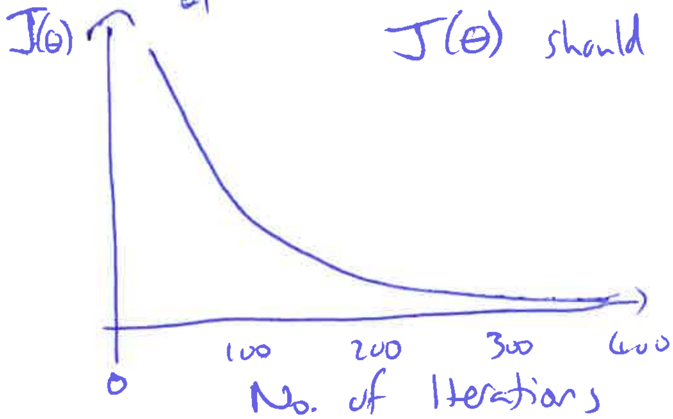
Also can do <u>mean normalization</u> i.e. $x_i := x_i - \mu_i$

so best: $x_i := \frac{x_i - \mu_i}{s_i} = \frac{x_i - (\text{mean of feature } i)}{(\text{Range of feature } i)}$

Making sure gradient descent is working correctly.

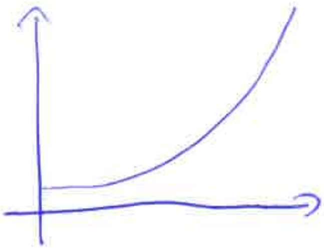
① Plot $J(\theta)$ on each iteration

$J(\theta)$ should decrease after every iteration.



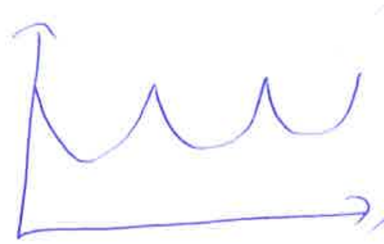
Declare convergence if $J(\theta)$ decreases by less than ε in one iteration? But hard to pick ε.

If



Use smaller α.

(or



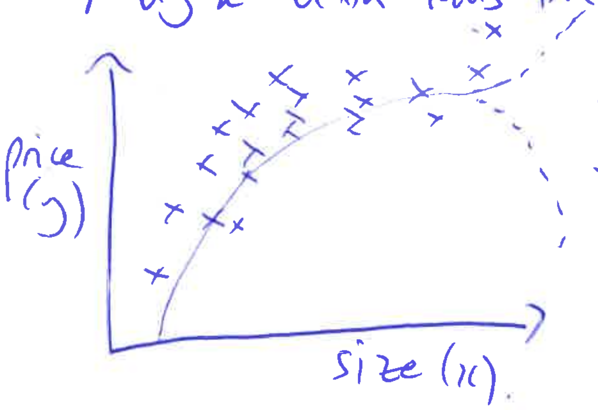
)

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

③

Polynomial Regression

Maybe data looks like:



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

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

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

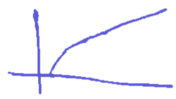
$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$

→ $x_1 = (size)$

$x_2 = (size)^2$ $\quad x_3 = (size)^3$

> NB If size: 1 - 1000
> $size^2$: 1 - 1000000
> $size^3$: 1 - 10^9
> So feature scale!

Other idea might be $h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 \sqrt{size}$



Normal Equation

Using Calculus we can solve the minimization of $J(\theta)$.

quick knowledge:

$$X = \begin{array}{c} \begin{array}{ccccc} x_0 & x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \end{array} \qquad y = \begin{array}{c} y \text{ (predict This)} \\ \begin{bmatrix} \vdots \end{bmatrix} \end{array}$$

$\boxed{\theta = (X^T X)^{-1} X^T y}$ Value of θ which minimises J.

More generally, m examples $(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})$; n features.

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\underset{\text{(design matrix)}}{X} = \begin{bmatrix} \text{---} (x^{(1)})^T \text{---} \\ \text{---} (x^{(2)})^T \text{---} \\ \vdots \\ \text{---} (x^{(m)})^T \text{---} \end{bmatrix} \quad m \times (n+1)$$

E.g. if $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$ $\quad \underline{X} = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_2^{(1)} \\ \vdots & \vdots \\ 1 & x_m^{(1)} \end{bmatrix}$ $\quad \underline{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

NB Feature scaling not required for normal equation.

m training examples, n features.

When to use Gradient Descent / Normal Equation

Gradient Descent	Normal Equation
• Need to choose α	• No need to choose α
• Needs many iterations	• Don't iterate
• Works well if n large	• Slow if n is very large
	• Need to compute $(X^TX)^{-1}$ — $n \times n$ matrix, $O(n^3)$ to do inverse.

NB $n = 1000$ is still relatively small.

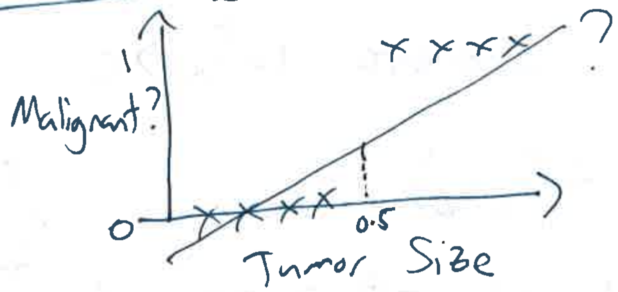
$n = 10^6 \quad \leftarrow \quad n = 10000$

① # Logistic Regression

Front.

Predicts a variable with classifications.

Example (using Linear regression)



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

Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict $y=1$.

Else $y=0$ (looks quite good).

But if $\exists$ an extra point, maybe has bad prediction?
So linear regression may not be the best, especially since we can have $h_\theta(x) > 1$ and $h_\theta(x) < 0$

Hypothesis Representation

Want $0 \leq h_\theta(x) \leq 1$.

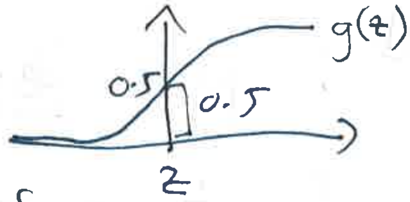
$h_\theta(x) = g(\theta^T x)$ where $g(z) = \frac{1}{1+e^{-z}}$ (g is the logistic or Sigmoid function)

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

Interpret $h_\theta(x) = p(y=1 \mid x;\theta)$ "probability that $y=1$, given x, parameterised by θ"

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

Decision Boundary



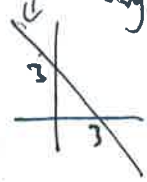
NB if I say $y=1$ if $h_\theta(x) \geq 0.5$

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

$h_\theta(x) = g(\theta^T x) \geq 0.5$ wherever $\underbrace{\theta^T x}_{z} \geq 0$

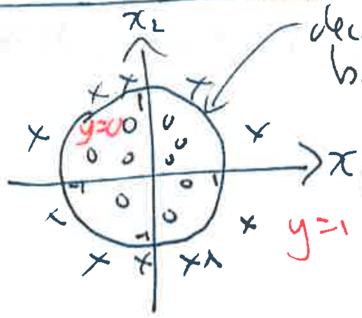
So. $h_\theta(x) = g(\underset{-3}{\theta_0} + \underset{1}{\theta_1} x_1 + \underset{1}{\theta_2} x_2)$

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



i.e. The decision boundary is a -1 hyperplane that separates the training data as best as possible. manifold?

Non-linear example



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

lets try $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$

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

Cost Function

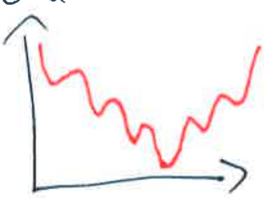
Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

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

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

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

NB using this cost function for Logistic regression is bad, since it would look like:

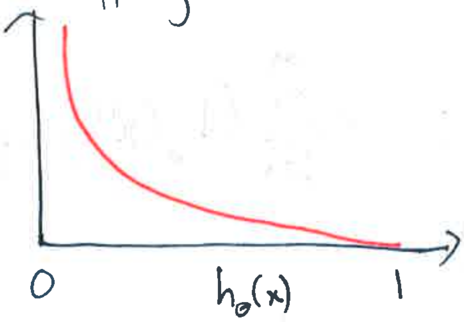


(not convex.

② front

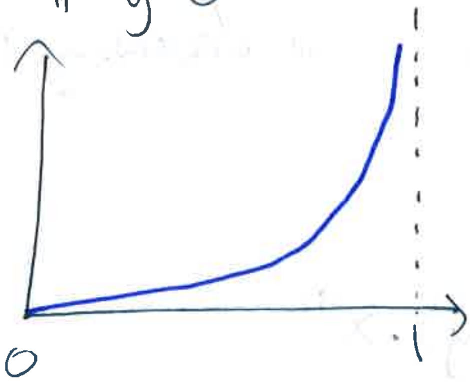
$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases} = \begin{cases} -\log\left(\frac{1}{1+e^{-\theta^T x}}\right) \\ \end{cases}$$

If $y=1$



Cost $= 0$ if $y=1$ and $h_\theta(x)=1$
But 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 a lot.

If $y=0$



I.E. high cost to algorithm if data says $y=0$, but we predict close to 1.

Recap (m = number of training examples).

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

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

NB $y=0$ or 1 always.

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

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

To fit parameters θ:

$$\min_\theta J(\theta)$$

to make a prediction given new x:

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

Gradient Descent

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

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

Repeat {

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

} (simultaneously update all θ_j).

Algorithm looks identical to liner regression!

Plot $J(\theta)$ on each iteration & check it is decreasing. (cost decreases).

Vectorized:

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

Advanced Optimisation

Given θ, we have code that can compute - $J(\theta)$, - $\frac{\partial}{\partial \theta_j} J(\theta)$ for $j = (0, 1, \ldots, n)$. to plug in to gradient descent.

other algorithms are:
- Conjugate gradient
- BFGS
- L-BFGS

Advantages: don't pick α, & faster than gradient descent.

Disadvantages: More complex.

Front

Example

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $\quad J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

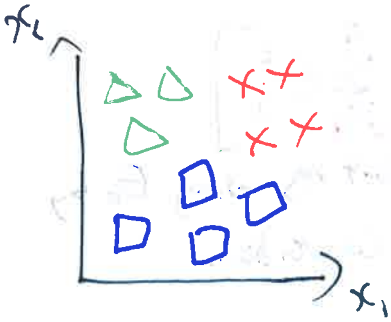
$$\frac{\partial}{\partial\theta_1} J(\theta) = 2(\theta_1 - 5), \quad \frac{\partial}{\partial\theta_2} J(\theta) = 2(\theta_2 - 5)$$

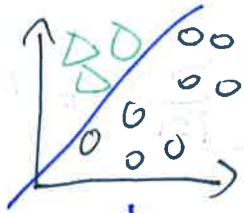
then we provide these to octave.

Multiclass classification

E.g. Email foldering : Work ($y=1$), Friends ($y=2$), Family ($y=3$), Hobby ($y=4$)

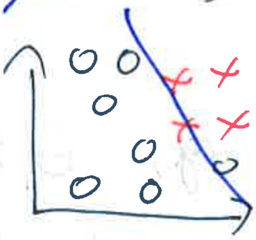
Weather : Sunny, Cloudy, rainy, fog





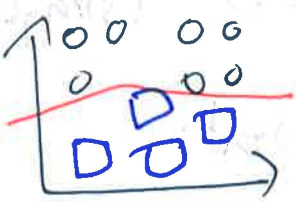
$h_\theta^{(1)}(x)$

$\leftarrow p(y=1 \mid x;\theta)$



$h_\theta^{(2)}(x)$

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



$h_\theta^{(3)}(x)$

On a new input x, to make a prediction, pick the class i that maximizes

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

Regularisation.

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

Idea:

Small values for parameters $\theta_0, \theta_1, \dots \theta_n$

- Simpler hypothesis
- Less prone to overfitting.

So use cost function: (linear regression).

regularisation parameter

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

don't reg using θ_0 by convention.

If λ very large, $\theta_1 \approx 0$, $\theta_2 \approx 0 \dots$ $\theta_n \approx 0$ and so $h_\theta(x) \simeq \theta_0$ (straight line)

New Gradient descent:

Repeat: {

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

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

}

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

NB: $1 - \alpha\frac{\lambda}{m} < 1$. $\alpha > 0$ small, so close-ish to 1.

So $\theta_j(1 - \alpha\frac{\lambda}{m}) < \theta_j$, so this shrinks θ_j slightly more than g.d. for normal linear regression.

front

Normal Equation

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

$\min_{\theta} J(\theta)$

$$\theta = \Big(X^T X + \lambda \underbrace{\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & & & 0 \\ 0 & 0 & 1 & & \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}}_{}\Big)^{-1} X^T y$$

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

Regularised Logistic Regression

Cost Function

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

Gradient Descent

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

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

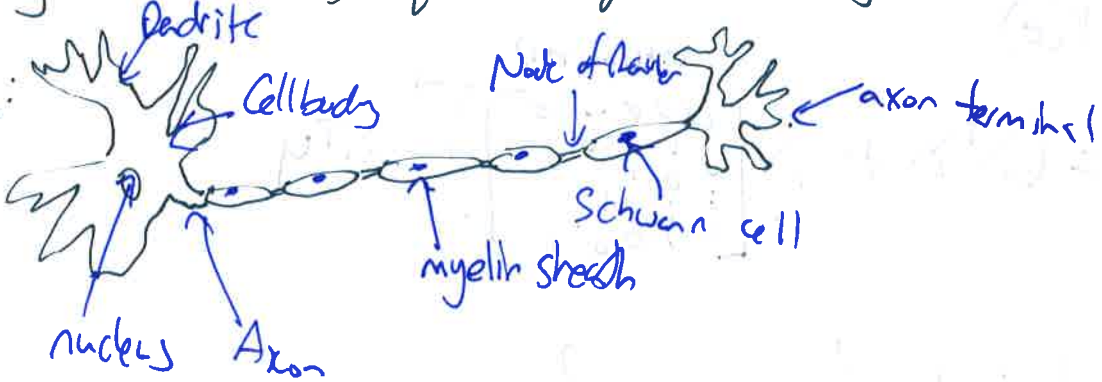
$j = 1, 2, \ldots, n$

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

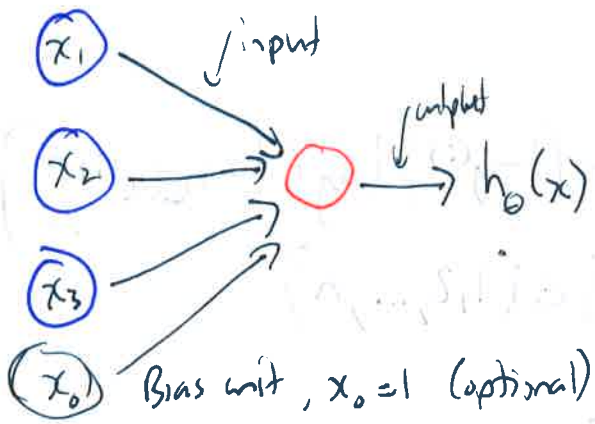
Neural Networks

If there is probably not a reasonable separation, in Linear & logistic regression, we would need high order polynomials which, with many features, is quite large! Overfitting / slow.

Neuron:



Neuron model: Logistic unit

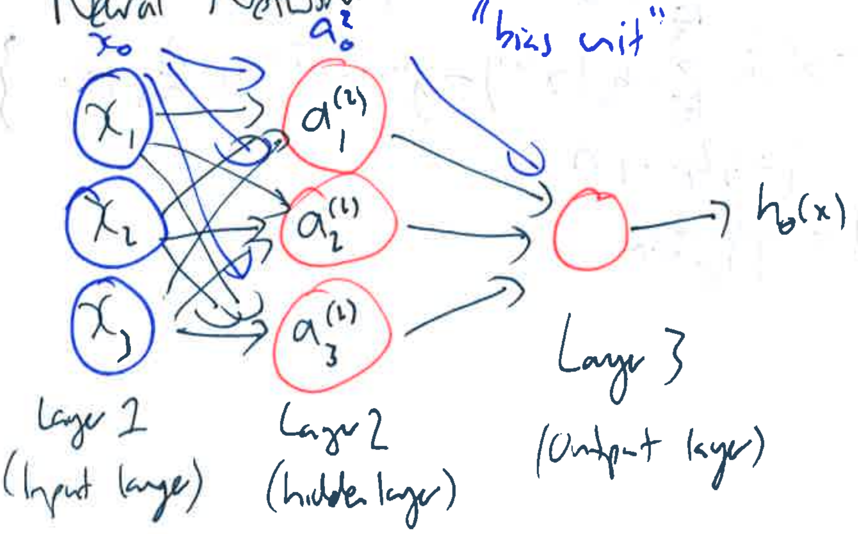


$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}, \quad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

weights of neural net model

Sigmoid (logistic) activation function.

Neural Network



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

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

Neural nets (2)

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

$$a_2^{(2)} = \dots$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) := g(z_3^{(2)})$$

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

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

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

Forward propagation: Vectorized implementation.

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

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

$:= \Theta^{(1)} a^{(1)}$

Add $a_0^{(2)} = 1. \rightarrow a^{(2)} \in \mathbb{R}^4$

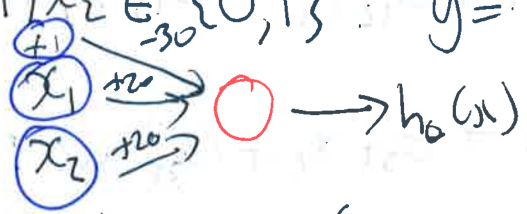
$\Rightarrow z^{(3)} = \Theta^{(2)} a^{(2)} \rightarrow h_\Theta(x) = a^{(3)} = g(z^{(3)})$.

Network Architecture: how net is connected.

hidden layers.

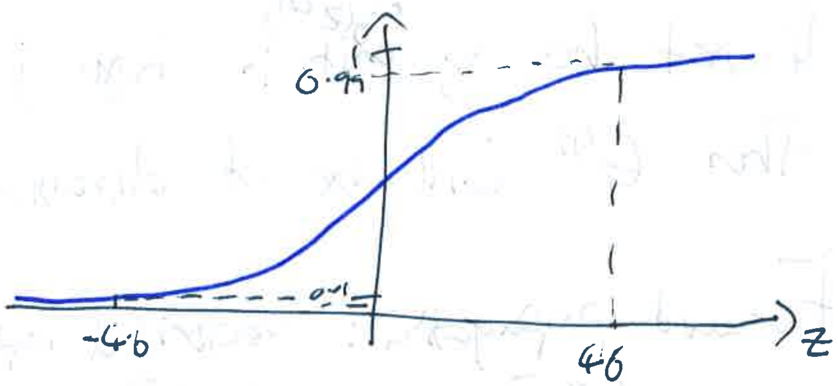
Example

AND $x_1, x_2 \in \{0,1\}$. $y = x_1 \text{ AND } x_2$

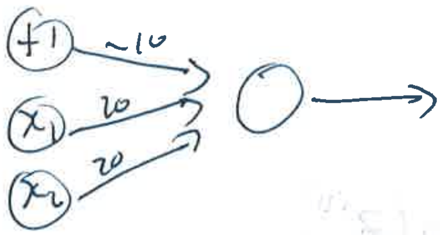


$$h_\theta(x) = g(-30 + 20x_1 + 20x_2).$$

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

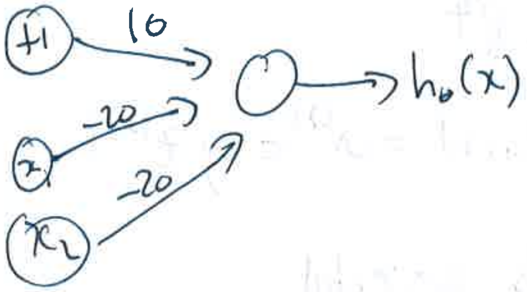


OR



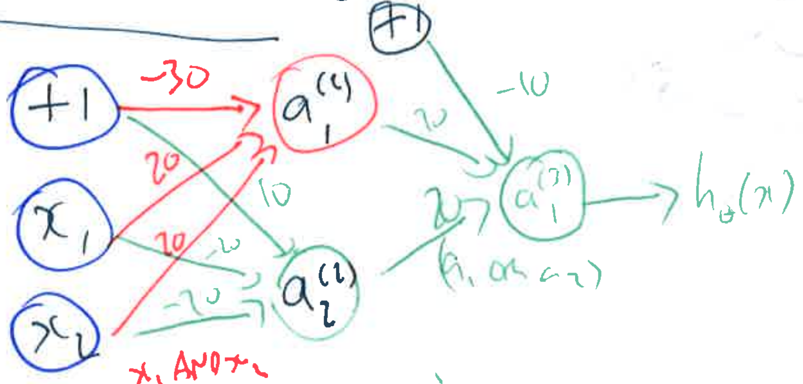
$y \approx x_1 \text{ OR } x_2$

Negation (not x_1 & not x_2)



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

x_1 XNOR x_2 (NOT (x_1 XOR x_2))



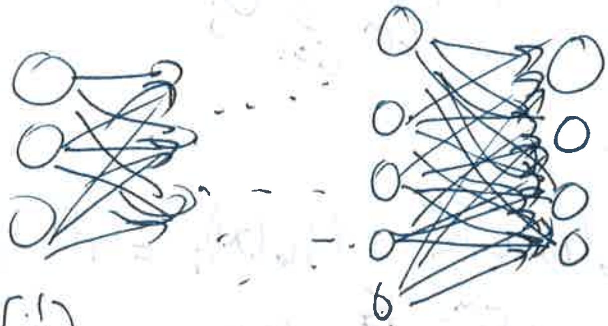
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Neural nets (3)

front

Multiple output units: One vs All.

e.g. Predict image is of pedestrian, Car, Motorcycle, Truck.



Want $h_\theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian, similar for others.

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

Notation

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

L = total no. of layers in network.

s_l = no. of units (excluding the bias unit) in layer l.

Binary Classification

$y = 0$ or 1

1 output units

$h_\theta(x) \in \mathbb{R}$

$s_L = 1 \quad k = 1$

↑ output units here

Multiclass classification (k classes)

$y \in \mathbb{R}^k$ e.g. $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

k output units

$h_\theta(x) \in \mathbb{R}^k$

$s_L = k \quad (k \geq 3)$

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:

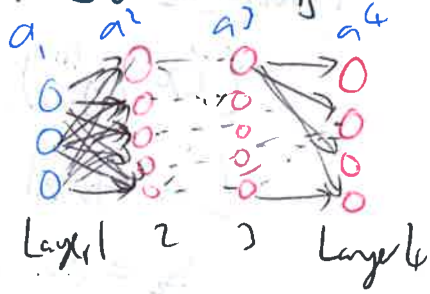
$h_\theta(x) \in \mathbb{R}^K \quad (h_\theta(x))_i = i^{th}$ output.

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

Gradient Computation

$\min_\Theta J(\Theta) \rightarrow$ Need code to compute: $J(\Theta)$ and $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

e.g. Given one training example (x, y):



Forward Propagation:

$a^{(1)} = x$

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

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

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

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

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

$a^4 = g(z^{(4)})$

Gradient computation: Backpropagation algorithm.

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

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

$\delta_j^{(4)} = a_j^{(4)} - y_j$ i.e. $= (h_\theta(x))_j - y_j$ (vectorized $\underline{\delta}^{(4)} = \underline{a}^{(4)} - \underline{y}$

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

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

Finally, you can prove

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)} \quad \text{(ignoring } \lambda\text{; if } \lambda = 0\text{).}$$

Example

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

Set $\Delta_{ij}^{(l)} = 0 \ \forall \ l, i, j$. (use to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta)$)

For $i = 1$ to m

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

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

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

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$.

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

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

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

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

Focusing on a single example $x^{(i)}, y^{(i)}$, 1 output unit and $\lambda = 0$,

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

(Think of $cost^{(i)} \approx (h_\theta(x^{(i)}) - y^{(i)})^2$)

(see slides on intuition).

Tip for advanced optimisation; they expect vectors, so can use [theta1(:); theta2(:)] to vectorize, &

theta1 = ~~thetavec~~ reshape(thetavec(1:100), 10, 10) e.g.

Gradient Checking

NB $\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$ try $\varepsilon \leq 10^{-4}$.

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

$\theta = \theta_1, \theta_2, \dots, \theta_n$.

e.g. $\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \dots, \theta_n) - J(\theta_1 - \varepsilon, \theta_2, \dots, \theta_n)}{2\varepsilon}$.

octave: for i = 1:n,

thetaPlus = theta

thetaPlus(i) = thetaPlus(i) + ε

thetaMinus = theta;

thetaMinus(i) = thetaMinus(i) − ε

GradApprox(i) = (J(thetaPlus) − J(thetaMinus))/(2ε);

end;

Check gradApprox ≈ DVec → from backProp. if so, implemented correctly.

Neural nets 5

Implementation.

(1) Implement backprop to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$)
(2) Implement gradient checking and ensure similar to (1).
(3) Use backprop & disable gradient checking.
(since v slow).

Random Initialisation

(1) initialTheta = zeros(n,1)? No! Since $a_1^{(2)} = a_2^{(2)}$ e.g.
and $\delta_1^{(2)} = \delta_2^{(2)}$, & $\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\theta)$

(2) Symmetry Breaking initialise each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$. e.g. Theta2 = rand(10,11) * (2 * init_Epsilon) − init_Epsilon.

Putting it together

(1) Decide on network architecture; # inputs, outputs, hidden layers & hidden units each.
Inputs = Dimension of features $x^{(i)}$
Outputs = Classes (recoded into binary variables).
Reasonable default: 1 hidden layer, or if > 1, same no. of hidden units in every layer (normally the more, the better).
normally # hidden units is 1 to 4 times num input units.

2 Training

a) Randomly initialize weights
b) Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$.
c) " code for cost function $J(\theta)$
d) Implement back prop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\theta)$.

for i = 1:m
Do forward prop & backprop for each example $x^{(i)}, y^{(i)}$
(Get activations $a^{(l)}$ & delta terms $\delta^{(l)}$ for $l = 2, \dots, L$.

e) Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\theta) \approx$ numerical estimate of $J(\theta)$.

f) Use gradient descent or adv. opt. method with back prop. to minimize $J(\theta)$.

What next? Practical Advice.

FRONT

① Suppose I implement regularized linear regression to predict housing prices.

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

However, when testing hypothesis on new data, $\exists$ a large error in predictions. What next?.

a) Get more training examples → fixes high variance
b) Try smaller set of features. → fixes high variance
c) Try getting additional features → Fixes high bias
d) Try adding polynomial features $(x_1^2, x_2^2, x_1x_2$ etc). fixes high bias
e) Try increasing/decreasing λ. fixes high variance.

But lots of options, maybe time consuming...

Machine learning diagnostic

Def^n: A test that you can run to gain insight on what does/doesn't work in your ML algorithm.

Evaluating your hypothesis: 70% Training, 30% test set.

a) Learn parameter θ from training data (minimizing training error $J(\theta)$).
b) Compute test set error:

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

Logistic Reg $J_{test}(\theta) = -\frac{1}{m_{test}}\sum_{i=1}^{m_{test}} y^{(i)}_{test}\log h_\theta(x^{(i)}_{test}) + (1-y^{(i)}_{test})\log h_\theta(x^{(i)}_{test})$

OR . Misclassification error:

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

Test error $= \frac{1}{m_{test}}\sum_{i=1}^{m_{test}} err(h_\theta(x^{(i)}_{test}), y^{(i)}_{test})$ (i.e. fraction wrong).

Model Selection

e.g. I try lots of different models, and calculate $J_{test}(\theta)$ for each, and try to minimize error on test set.

Problem: $J_{test}(\theta^{(5)})$ (say) is likely to be an optimistic estimate of generalization error. i.e. our extra parameter (d = degree of polynomial) is fit to the test set.

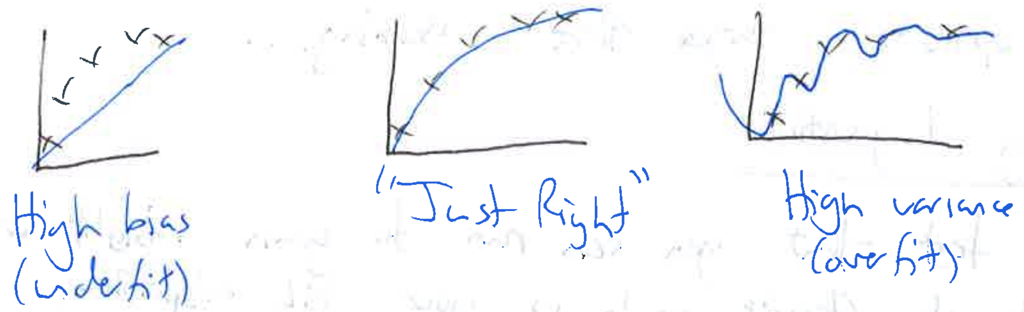
Better split! 60% Training, 20% cross-validation (cv). $(x^{(i)}_{cv}, y^{(i)}_{cv})$, 20% test.

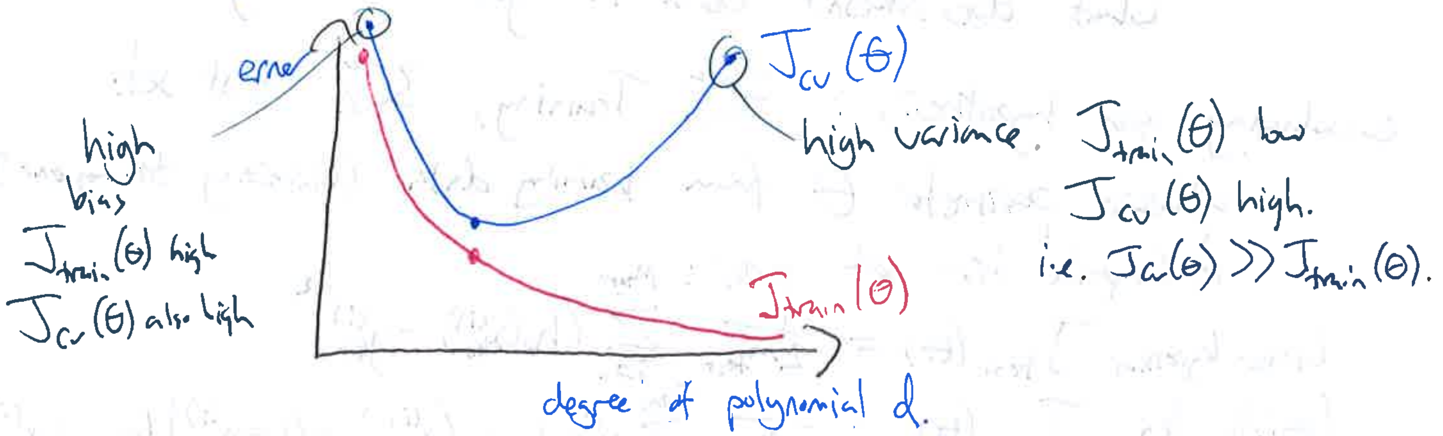
Now on train: $\min_\theta J(\theta)$ over each model.

Calculate $J_{cv}(\theta)$ for each model. say $J_{cv}(\theta^{(4)})$ the best.

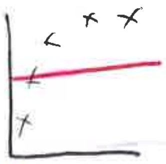
Calculate generalization error for test set $J_{test}(\theta^{(4)})$.

Bias & Variance





Bias & Variance 2 (Regularization).



Large λ

High Bias (underfit)

$\lambda = 10000$ $\theta_1 \approx 0, \theta_2 \approx 0, \ldots$

$h_\theta(x) \approx \theta_0$

Intermediate λ

Small λ

High variance (overfit)

$\lambda \approx 0$.

Choosing λ

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

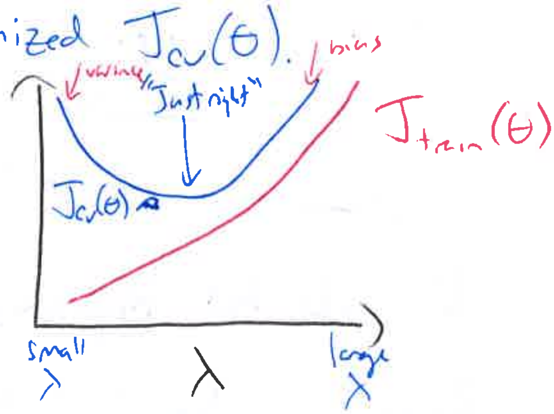
Try $\lambda = 0, 0.01, 0.02, 0.04, 0.08, \ldots, 10.24$.

(1), (2), (3), (4), (5), …, (12)

Calculate each $\min_\theta J(\theta)$ for each model, $\theta^{(1)}$, then calculate $J_{cv}(\theta)$ for each model & pick λ according to minimized $J_{cv}(\theta)$.

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

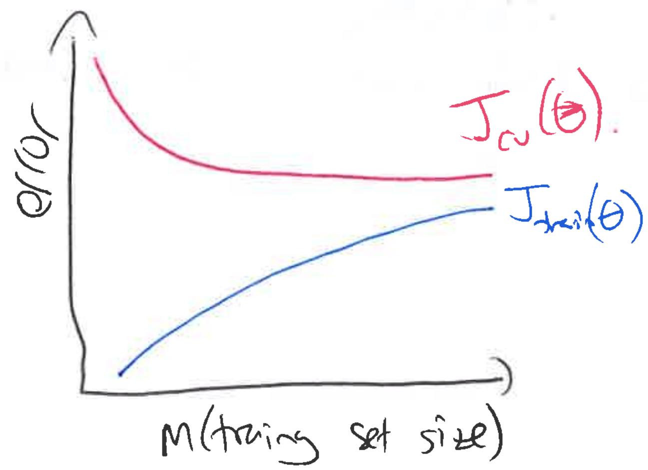
$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)}\right)^2$$

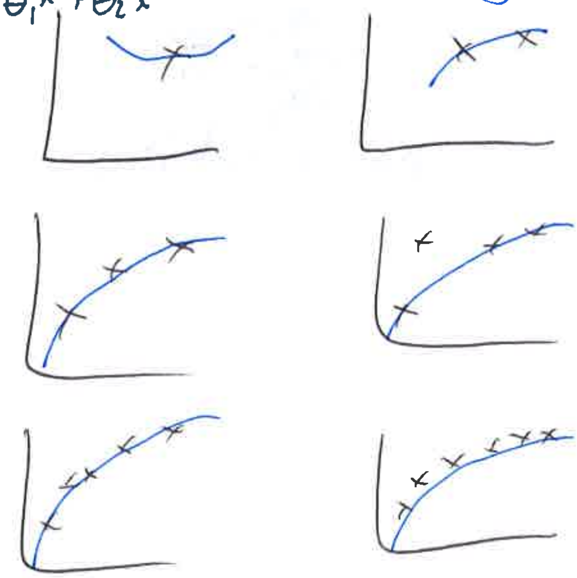


Learning curves

$J_{train}(\theta)$ and $J_{cv}(\theta)$ as above. Use small # of training examples. (10 or 20).

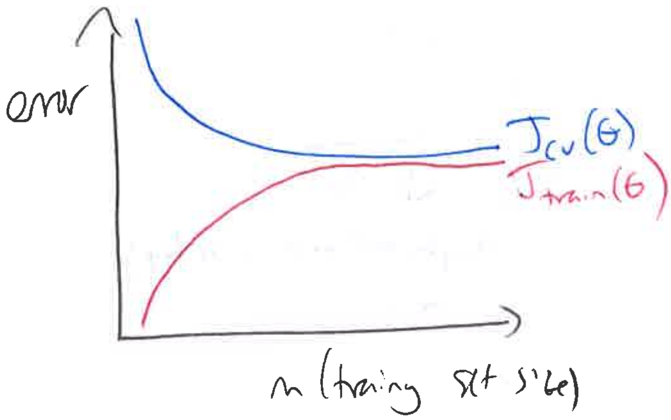
$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

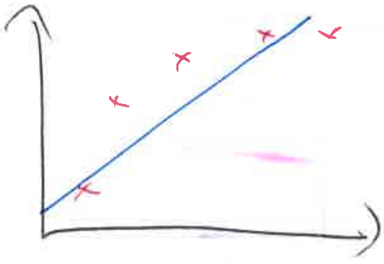




High Bias

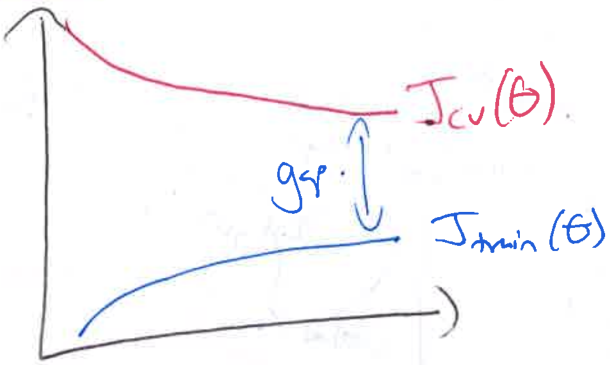
$$h_\theta(x) = \theta_0 + \theta_1 x$$

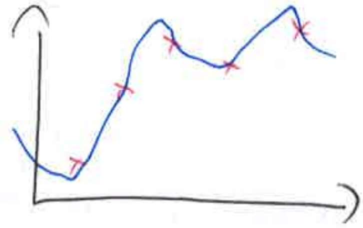


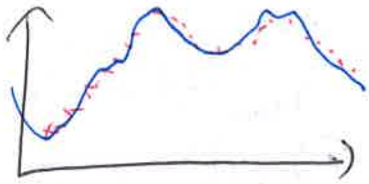


If model has high bias, more data will not help!

High Variance







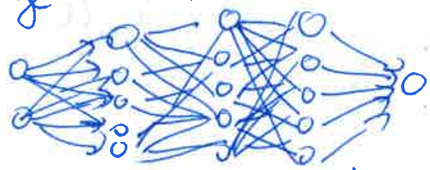
If model has high variance, getting more training data is likely to help.

Briefly on Neural Nets

"Small" Neural net

(prone to underfitting)

"Large" Neural Net



Use regularisation to address overfitting.

Building a Spam Classifier. front

Supervised Learning.
x = features of email. y = Spam (1) or not spam (0)
Features x: Choose 100 words indicative of spam / not spam.
For each word, test for existence of word in email. x then a vector like $\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$. (but $x \in \mathbb{R}^{100}$).

i.e. $x_j = \begin{cases} 1 & \text{if word } j \text{ appears in email} \\ 0 & \text{otherwise.} \end{cases}$

Reduce error by collect lots of data (honeypot project).
Develop sophisticated features based on email routing information (email header)
Stemming? Misspellings?
↳ "Porter Stemmer" software.

New Model Approach

- Quick & Dirty – do a simple algorithm that can be implemented quickly.
- Plot learning curves to decide if more data / features etc. are likely to help.
- Error Analysis: Manually examine examples in the CV set that the algorithm made errors on. Spot a systematic patterns in what type of example it makes errors on.
  e.g. in spam email example what types are miscategorised, e.g.
  Pharma (12), replicas (4), Steal passwords (53). Focus on 53, maybe most have odd punctuation? So focus on that.

Skewed classes may give a problem!

Precision / Recall

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

		Actual 1	Actual 0
predicted	1	True positive	false positive
	0	false negative	True negative

Precision

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

$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positive}}{\text{True pos} + \text{false pos}}$$

Recall

(Of all the patients that actually have cancer, what fraction did we correctly detect as having cancer?)

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

Trading off precision and recall

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

Predict 1 if $h_\theta(x) \geq 0.5$, 0 otherwise.

Suppose we want to predict $y=1$ (cancer) if very sure.

One way, change limit of 0.5 to 0.7, say.

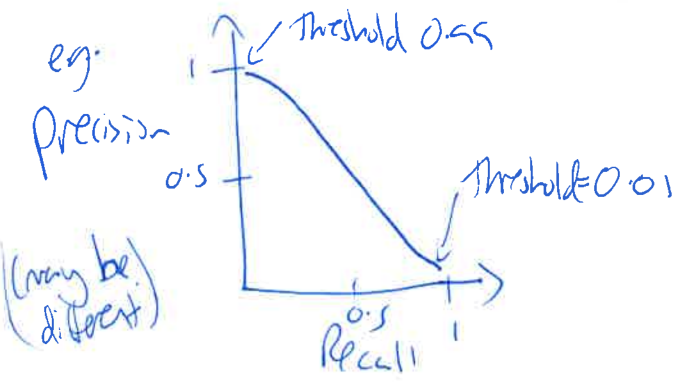
$\Rightarrow$ Higher precision, lower recall.

Suppose we want to avoid missing too many cases of cancer. We then try limit of 0.3

$\Rightarrow$ Lower precision, higher recall.

More generally: Predict 1 if $h_\theta(x) \geq$ threshold.

eg.



front

How to pick threshold?

F_1 Score (F score).

	Precision (P)	Recall (R)	~~Average~~	F_1 Score
Algorithm 1	0.5	0.4	~~0.45~~	0.444
Algorithm 2	0.7	0.1	~~0.4~~	0.175
Algorithm 3	0.02	1.0	~~0.51~~	0.0392

BAD

$$\text{Average} = \cancel{\frac{P+R}{2}} \qquad F_1 \text{ Score} = 2\frac{PR}{P+R}$$

NB if $P=0$ or $R=0$ $\Rightarrow$ F-score $=0$ ✓

if $P=1$ and $R=1$ $\Rightarrow$ F-score $=1$ ✓

Now When do I want more data?

Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately. E.g. For breakfast I ate ~~to~~ ~~too~~ two eggs.

Counter example: Predict housing price from size (feet^2) only.

Useful test: Given input x, can a human expert confidently predict y?

Use a learning algorithm with many parameters. (e.g. many features/hidden units). $\boxed{\text{Low Bias Algorithms}}$ $\Rightarrow J_{train}(\theta)$ will be small.

If I use a very large training set (unlikely to overfit). $\boxed{\text{low variance}}$

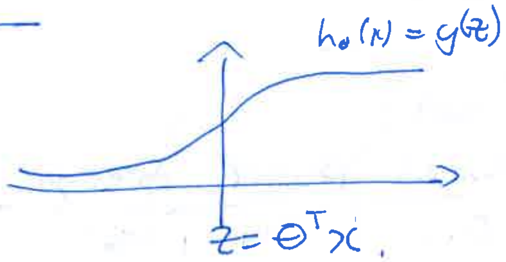
$\rightarrow J_{train}(\theta) \approx J_{test}(\theta)$

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

Support Vector Machines

log reg

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

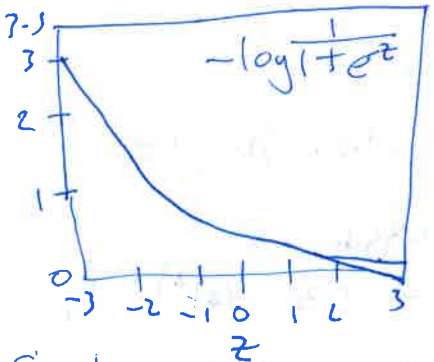


If $y=1$, we want $h_\theta(x) \approx 1$, $\theta^T x \gg 0$.
If $y=0$, we want $h_\theta(x) \approx 0$, $\theta^T x \ll 0$.

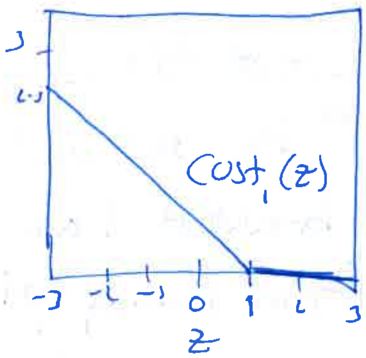
Cost of an example: $-(y \log h_\theta(x) + (1-y)\log(1-h_\theta(x)))$

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

If $y=1$ (want $\theta^T x \gg 0$):



new cost func for SVM.



Similar for $y=0$. and get $cost_0(z)$.

SVM: lose $\frac{1}{m}$ from log res, just convention!

$$\min_\theta \underbrace{C\left[\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)})(cost_0(\theta^T x^{(i)})\right]}_{A} + \underbrace{\frac{1}{2}\sum_{j=0}^{n} \theta_j^2}_{B}$$

$\min_\theta \underline{A} + \lambda \underline{B}$

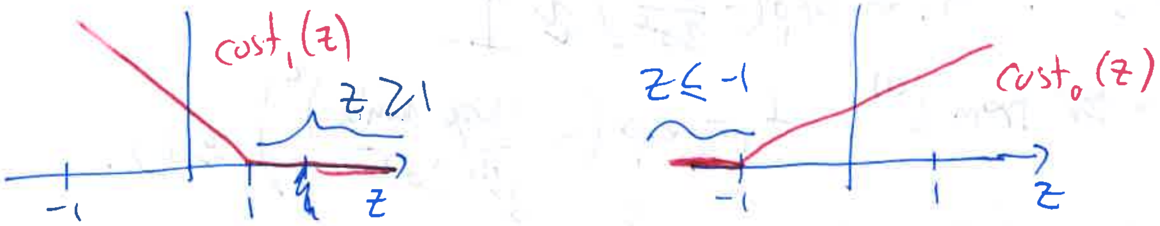
but actually by convention, $\min_\theta c\underline{A} + \underline{B}$ if $c = \frac{1}{\lambda}$ then same.

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

front.

Recap

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} \, cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) \, cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$



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

So if we try C very big

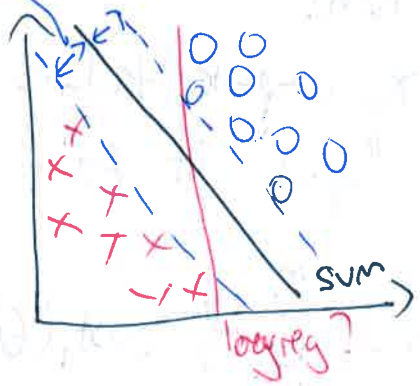
Wherever $y^{(i)} = 1$:
$\theta^T x^{(i)} \geq 1$

$$\min C \times 0 + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$
$$\text{s.t. } \theta^T x^{(i)} \geq 1 \text{ if } y^{(i)} = 1$$
$$\theta^T x^{(i)} \leq -1 \text{ if } y^{(i)} = 0.$$

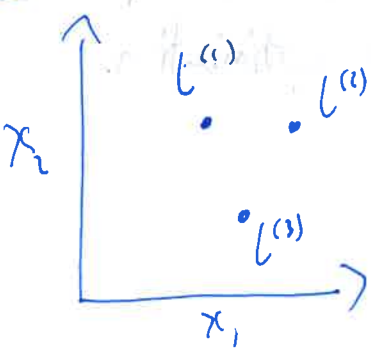
Wherever $y^{(i)} = 0$:
$\theta^T x^{(i)} \leq -1$

More robust since requires a large margin for separation.

NB if C very large, very sensitive to outliers.



Kernel



Given x, compute new feature depending on proximities to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$.

kernel (Gaussian)

Given x: $f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$
$f_2 = \text{similarity}(x, l^{(2)})$
$f_3 = \text{similarity}(x, l^{(3)}) = k(x, l^{(i)})$

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

If $x \approx l^{(1)}$: $f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$.

If x is far from $l^{(1)}$: $f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$.

NB graph is like a bump.

Smaller σ = narrower bump.

Predict $y=1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$.

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?

Use each training example as a landmark! i.e. $l^{(i)} = x^{(i)}$ for $i = 1 \ldots m$

Given x: $f_1 = \text{similarity}(x, l^{(1)})$ $\quad x^{(i)}$

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

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

For a training example: $x^{(i)} \rightarrow$

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

$\vdots$

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

$x^{(i)} \in \mathbb{R}^{n+1}$ (or $\mathbb{R}^n$)

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$$

Usage with SVM:

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

Predict "$y=1$" if $\theta^T f \geq 0$ nb $\theta^T f = \theta_0 f_0 + \ldots + \theta_m f_m$

Training

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

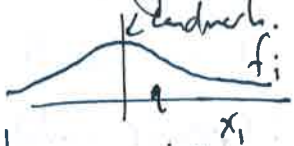
NB actually last term is $\theta^T \theta$, but implemented as $\theta^T M \theta$ for some matrix M for optimisation.

front

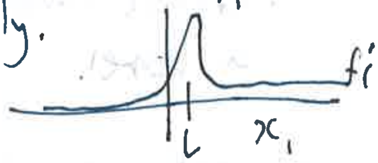
SVM Parameters

$C (= \frac{1}{\lambda})$. Large C: Lower bias, high $\overset{\text{overfit}}{\text{variance}}$ (small λ).
Small C: Higher bias, low variance (large λ).

σ^2: Large σ^2: Features f_i vary more smoothly.
Higher bias, lower variance.



Small σ^2: Features f_i vary less smoothly.
Lower bias, higher variance.



NB Software packages for SVMs. e.g. liblinear, libsvm to solve for parameters Θ.

Need to specify C & choice of kernel (similarity function):

e.g. No kernel ("linear kernel") = predict "$y=1$" if $\Theta^T x \geq 0$
↑ do this if n large, m small.

Gaussian kernel (as we just saw). Must choose σ^2.
↑ if n small, m large?

If Gaussian:

~~Handout~~ function $f_i = \text{kernel}(\overset{x^{(i)}}{x1}, \overset{l^{(j)}}{x2})$

$$f_i = \exp\left(-\frac{\|x1 - x2\|^2}{2\sigma^2}\right)$$

return.

NB: Do perform feature scaling before using Gaussian kernel.

Note: Not all similarity functions make valid kernels (Need to satisfy "Mercer's Theorem").

Recall One-vs-all method (Train k models, one to distinguish $y=i$ from the rest). (pick class i with largest $(\Theta^{(i)})^T x$).

Logistic Regression vs SVMs.

If n is large (relative to m): ($n \geq m$, $n = 10{,}000$ $m = 10-1000$).
use log. reg. or SVM with no (linear) kernel.

If n is small, m is intermediate ($n = 1-1000$, $m = 10-10{,}000$)
use SVM with gaussian kernel

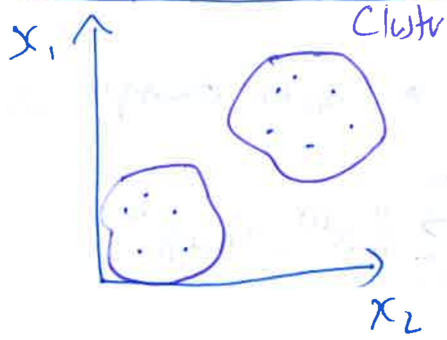
If n is small, m large ($n = 1-1000$, $m = 50000+$)
: Create/add more features, then use log reg. or SVM without a kernel.

Neural nets should do well for all of these, but might be slow.

F

Unsupervised Learning

Clustering Algorithm.



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

Applications of clustering.

1. Market Segmentation.
2. Social Network Analysis.
3. Organize Computing Clusters
4. Astronomical Data Analysis.

K-Means Algorithm

1) Randomly create cluster centroids (may not be in training set).
2) Assign each training example to a cluster (closest centroid).
3) Move cluster centroids by average location in each cluster.

until converge.

Inputs: - K (number of clusters).

- Training set $\{x^{(1)}, \dots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention).

More Rigorous:

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$.

Repeat { for $i = 1$ to m

$c^{(i)} :=$ index (from 1 to K) of cluster centroid closest to $x^{(i)}$. $\quad c^{(i)} = \min_k \|x^{(i)} - \mu_k\|^2$

for $k = 1$ to K

$\mu_k :=$ average (mean) of points assigned to cluster k.

}

$$\mu_k = \frac{1}{|N|} \sum_{m \in \{i : c^{(i)} = k\} := N} x^{(m)}$$

If centroid has no points assigned to it, either discard, or randomly initialise again.

Optimisation Objective

Notation: $\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned.

Opt. Obj: $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2$

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

NB in our algorithmic design, we try to min J wrt $c^{(1)}, \ldots, c^{(m)}$ and then wrt $\mu_1, \ldots, \mu_K$.

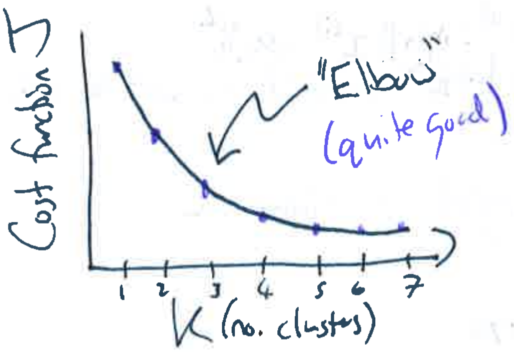
Random Initialisation

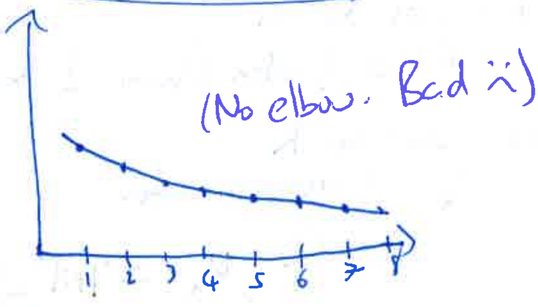
① Obviously $K < m$. ② Randomly pick K training examples.
③ Set $\mu_1, \ldots, \mu_K$ to be equal to these K examples.

Note, we can get stuck at bad local optima... So!
Run k-means 50-1000 times. Pick the clustering solution with the lowest cost $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$.
(More likely to have a big benefit if $K = 2, \ldots, 10$).

Choosing the number of Clusters (K)

Not always a clear cut answer! Elbow Method:



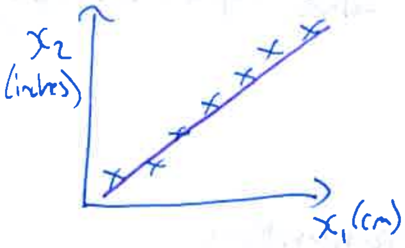


Alternatively, evaluate based on how it performs for later purpose, i.e. think about the business reason for doing this.

F

Dimensionality Reduction

Motivation I: Data Compression



Reduce data from 2D to 1D.
Do this by projecting examples onto line & measure distance along this line.

Motivation II: Visualisation

Get a 2D/3D Viz to better understand data.

Principal Components Analysis (PCA)

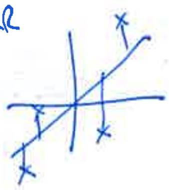
Problem Formation: Try to find a surface to project data to which minimizes projection distance.

easy: Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^2$) onto which to project the data as to minimize the projection error.

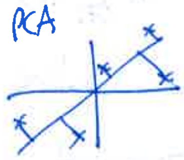
General: Reduce from n-dimensions to k-dimensions: Find k vectors $u^{(1)}, \dots, u^{(k)}$ onto which to project the data so as to minimize the projection error. NB. Projecting onto the linear subspace spanned by $u^{(1)}, \dots, u^{(k)}$.

Note: PCA is not linear regression. LR. minimises "vertical" distance. PCA is "closest":

LR



PCA



Algorithm

Pre-processing: Training set $x^{(1)}, \dots, x^{(m)}$. $\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$. Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

Then scale features to have comparable range of values.

i.e. $x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$ $\leftarrow$ max-min OR stddev.

To reduce data from n-dimensional to k-dimensional.

$X = \begin{bmatrix} - x^{(1)T} - \\ \vdots \\ - x^{(m)T} - \end{bmatrix}$

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^{n} \underbrace{(x^{(i)})}_{n\times 1} \underbrace{(x^{(i)})^T}_{1\times n}$$ so Σ is $n \times n$. in octave, Sigma $= \frac{1}{m}$ * X' * X;

Compute eigenvectors of matrix Σ:

(in octave)

[U,S,V] = svd(Sigma); (singular value decomposition).

U is $n \times n$ $= \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(m)} \\ | & | & & | \end{bmatrix}$ $U \in \mathbb{R}^{n \times n}$. Take first k $u^{(i)}$'s for the projection we need.

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

$$z = \underbrace{\left[u^{(1)}, u^{(2)}, \ldots, u^{(k)} \right]}_{n \times k}^T \underbrace{x}_{n\times 1} = k \times 1 \text{ vector.}$$

Ureduce

Ureduce = U(:, 1:k);

z = Ureduce' * x

Choosing k (number of principal components)

Average squared projection error $= \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - x^{(i)}_{approx} \|^2$

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

Typically choose k to be smallest value st: $\frac{\text{error}}{\text{variation}} \leq 0.01$

$\Rightarrow$ 99% of variance is retained. (some use 0.05).

How? Try $k=1$. check if $\frac{\text{error}}{\text{variation}} \leq 0.01$. If not try $k=2$ etc.

OR [U,S,D] = svd(Sigma).

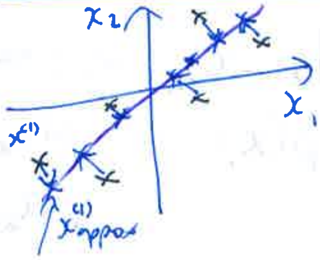
$S = \begin{bmatrix} s_{11} & & & & 0 \\ & s_{22} & & & \\ & & s_{33} & & \\ & & & \ddots & \\ 0 & & & & s_{nn} \end{bmatrix}$ For given k, $\frac{\text{error}}{\text{variation}} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$

So can test $\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$.

F

PCA: reconstruction from compressed representation

Recall, $z = U_{reduce}^T x$.



$X_{approx} = U_{reduce} z$

PCA: Application

If large number of features, our learning algorithm might be small. for a supervised problem.

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

Extract $x^{(i)}$'s $\in \mathbb{R}^{10000}$ (say)

$\downarrow$

$z^{(i)}$'s $\in \mathbb{R}^{1000}$ (say).

New training set $(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})$.

For new x, use same mapping as used before to get z. i.e. use same mapping as found on the training set. use for $x_{cv}^{(i)}$ and $x_{test}^{(i)}$

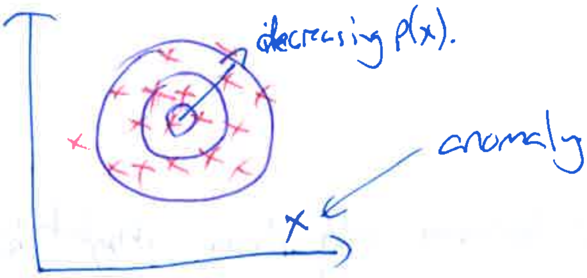
NB. Don't use to address overfitting; regularisation is much better. This is due to PCA losing information with no regard to y.

NB: Try without PCA first if possible! Only use if you don't get the desired output / too slow / not enough disk space.

Anomaly Detection

Dataset : $\{x^{(1)}, \dots, x^{(m)}\}$. Is x_{test} anomalous?

We will create a model $p(x)$: $p(x_{test}) < \varepsilon \rightarrow$ flag anomaly
$p(x_{test}) \geq \varepsilon \rightarrow$ OK



Fraud detection!

$x^{(i)}$ = features of user i's activities.

Model $p(x)$ from data.

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

use: Monitoring computers in a data center.

$x^{(i)}$ = features of machine i.

x_1 = memory use, x_2 = # disk accesses/sec.

x_3 = CPU load, x_4 = CPU load/network traffic.

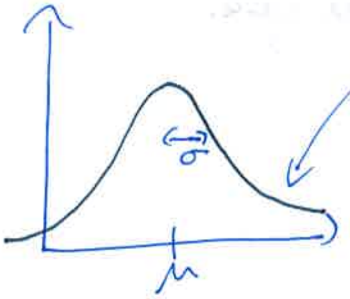
Gaussian/Normal distribution

Say $x \in \mathbb{R}$. If x is a distributed Gaussian with mean μ, variance σ^2

$x \sim \mathcal{N}(\mu, \sigma^2)$

parameterized by

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



Estimating parameters : Dataset $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ $x^{(i)} \in \mathbb{R}$. Say I suspect $x^{(i)} \sim \mathcal{N}(\mu, \sigma^2)$. Then $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2$ (can use $\frac{1}{m-1}$ doesn't matter)

Anomaly Detection Algorithm

Training set $\{x^{(1)}, \dots, x^{(m)}\}$, each $x \in \mathbb{R}^n$.

$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$
$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$
$\vdots$
$x_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$

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

(NB works fine even if x_i are not all independent).

$= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$ (Density Estimation).

① Choose features x_i that you think might be indicative of anomalous examples.

② Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

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

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

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

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

Anomaly if $p(x) < \varepsilon$.

Developing an anomaly detection system

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

Training set $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ (assume normal (not anomalous) examples).

Cross validation $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
test set $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$
} some $y=1$.

Aircraft example

10,000 good engines
20 flawed engines. (normally 2-50) ($y=1$).

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

Evaluating Algorithm

Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$.

On a cross validation/test example x, predict:

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

Possible Evaluation metrics:

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

Can also use CV set to choose parameter ε.

Anomaly Detection vs Supervised Learning

Anomaly Detection	Supervised Learning.
Very small number of positive examples ($y=1$). (0-20 is very common). Large number of negative ($y=0$) examples.	• Large number of positive and negative examples.
• Many different "types" of anomalies. Hard for algorithm to learn from positive examples look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.	• Enough positive examples for algorithm to get a sense of what positive examples look like, future positive examples likely to be similar to the ones in training set.
☐ Fraud detection e.g. loads of online retail fraud	☐ Email spam classification.
☐ Manufacturing (e.g. aircraft engines)	☐ Weather prediction
☐ Monitoring machines in a data center.	☐ Cancer classification.
⋮	⋮

F)

Anomaly Detection – Feature selection

① Plot data to check if it looks Gaussian. Often it's ok if not... octave command hist. What if poisson? Log transform!

other choices: $x_1 \leftarrow \log(x_1)$

$x_2 \leftarrow \log(x_2+1)$

$x_3 = x_3^{1/2}$

$x_4 = x_4^{1/4}$.

– Error analysis for anomaly detection.

Want $p(x)$ large for normal examples x.

$p(x)$ small for anomalous examples x.

Most common problem: $p(x)$ is comparable (say large) for both normal & anomalous examples. If so, take example, what went wrong? Inspire for new features.

Multivariate Gaussian Distributions

F

Recommender Systems

E.g. Amazon / Netflix recommend things you might like.

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at Last	5	5	0	0
Skyfall	5	?	?	0
Goldeneye	?	4	0	?
Car chase	0	0	5	4
Missing Kate	0	0	5	?

↑ stars 0-5.

n_u = # of users

n_m = no. movies.

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

$y^{(i,j)}$ = Rating given by user j to movie i

$n_u = 4$, $n_m = 5$

Learning algorithm must predict ~~...~~ $y^{(i,j)}$ if unassigned.

Content based recommender system

Start by rating categorys of movies. e.g. as before,

x_1 romance	x_2 action
0.9	0
1.0	0.01
0.99	0
0.1	1.0
0	0.9

$n = 2$

$$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix} \leftarrow x_0$$

$x^{(1)}$ is features of movie 1.

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

↑ like linear regression.

$m^{(j)}$ = no. of movies rated by user j.

To learn $\theta^{(j)}$:

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

To learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$:

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

(lost $\frac{1}{m^{(j)}}$ to make math easier).

Gradient descent update

if $k \neq 0$, add $+\lambda\theta_k^{(j)}$

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

F] Large Scale Machine Learning

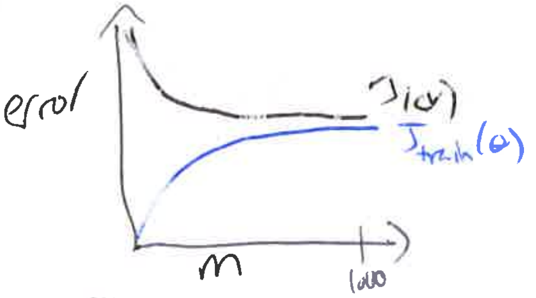
Why want a lot of data?

E.g. {to, two, too}, {then, than} confusable words.

Learning with large datasets ($m = 100,000,000$)

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

First try $m = 1,000$, say, and if:



(high bias)

then probably bigger dataset won't help

Reminder: Batch Gradient Descent for Linear Regression:

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

Repeat {

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

$\forall j \in \{0, 1, \ldots, n\}$

}

Stochastic Gradient Descent:

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

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

1. Randomly shuffle dataset.
2. Repeat { for $i = 1, \ldots, m$ { $\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) . x_j^{(i)}$ (for $j = 0, \ldots, n$)}}

1 to 10 times

Idea → each iteration fits to one example only.

So far:

Batch GD: Use all m examples in each iteration
Stochastic GD: Use 1 example in each iteration

Now!

Mini-batch GD: Use b examples in each iteration.
where $b =$ mini-batch size. $b = 2$-100 normally

E.g. Get $b=10$ examples $(x^{(i)}, y^{(i)}), \ldots, (x^{(i+9)}, y^{(i+9)})$

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

then do next 10. .

Say $b=10$, $m=1000$:

Repeat {
  for $i = 1, 11, 21, \ldots, 991$ {
    $$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$
    (for every $j = 0, \ldots, n$)
  }
}.

Mini-batch will outperform Stochastic GD if you use a good vectorization (allows for parallelisation).

F) Checking for convergence

Batch GD:

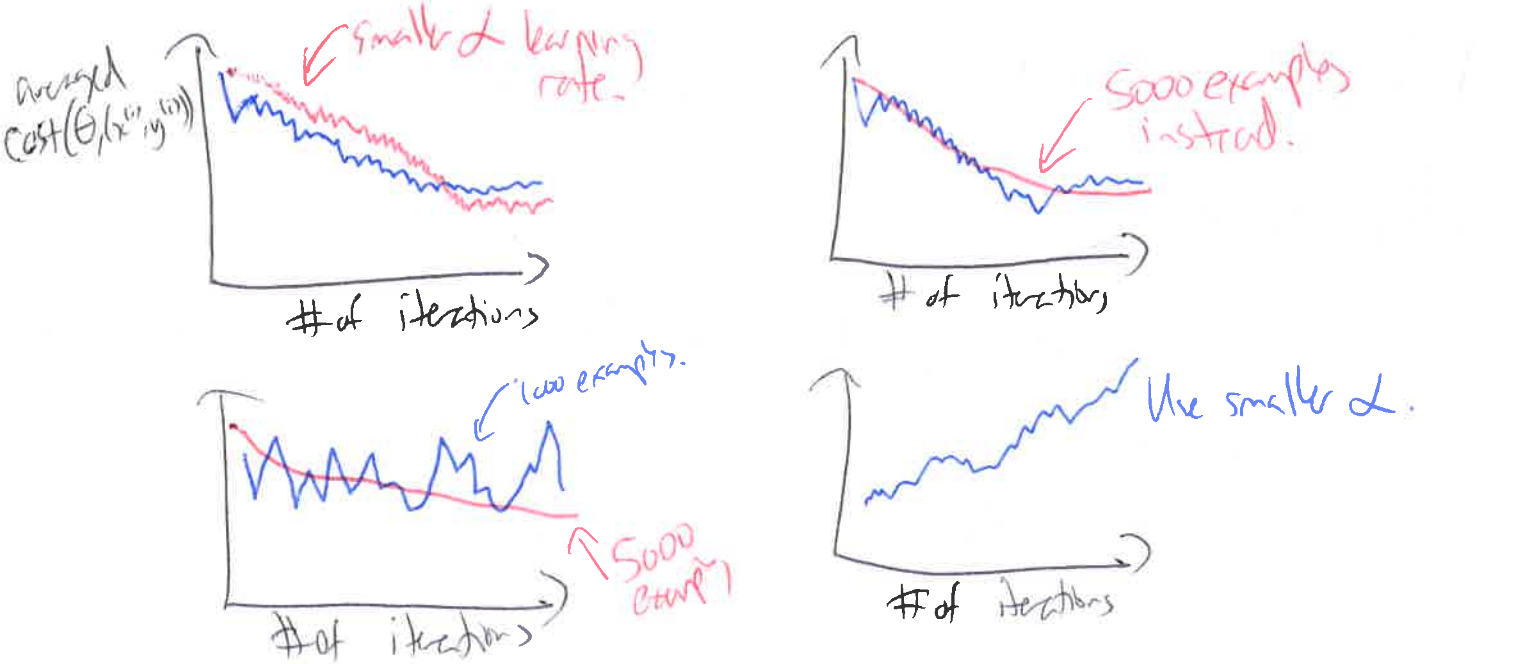
Plot J_{train} as a function of the # of iterations of gradient descent.

Stochastic GD:

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

During learning, compute $cost(\Theta, (x^{(i)}, y^{(i)}))$ before updating Θ using $(x^{(i)}, y^{(i)})$.

Every 1000 iterations (say), plot $cost(\Theta, (x^{(i)}, y^{(i)}))$ averaged over the last 1000 examples of algorithm. See possible examples:



To help stochastic GD converge, slowly decrease α over time.

e.g. $\alpha = \frac{const1}{iterationNumber + const2}$.

Online Learning

Suppose I offer a shipping service and users choose to use it based on price ($y=1$) or not use it ($y=0$).

Features x capture properties of user, of origin / destination and asking price. We want to learn $p(y=1|x;\theta)$ to optimize price. Lets use logistic regression.

Repeat forever { Get (x,y) corresponding to user.
Update θ using (x,y):
$$\theta_j := \theta_j - \alpha(h_\theta(x)-y)x_j \quad (j=0,\dots,n)$$
}

NB this can adapt to changing user preferences.

Map Reduce

Say I am using batch gradient descent, and $m = 400(,000,000)$.

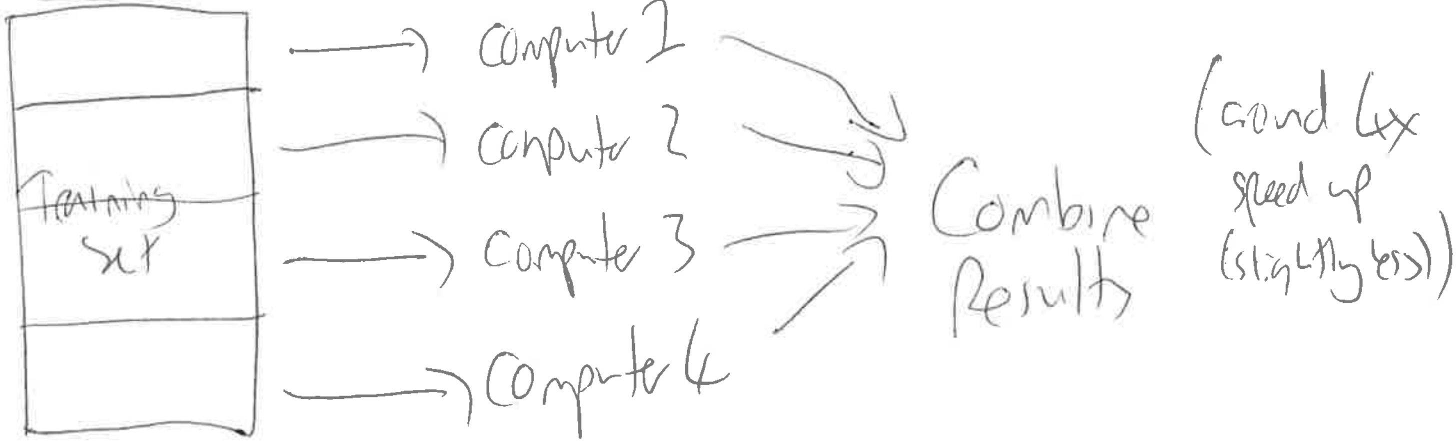
Machine 1: Use $(x^{(1)},y^{(1)}),\dots,(x^{(100)},y^{(100)})$
Compute $temp_j^{(1)} = \sum_{i=1}^{100}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$

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

etc. for Machine 3 & 4.

then: $\theta_j := \theta_j - \alpha\frac{1}{400}(temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$
$j=0,\dots,n$.

F) Map reduce



So we can use Map Reduce when doing very large sums. NB: Sometimes multiple cores can help on a single computer!