with some manipulation our update rule can also be represented as:

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

The first term in the above equation, $\left(1-\alpha\frac{\lambda}{m}\right)$ will be less than 1. intuitively you can see it as reducing the value of θ_j by some amount on every update. Notic that the second ~~the~~ term is now exactly the same as ~~be~~ it was before.

★ Normal Equation

Now lets approch regularization using the alternate methode of non-iterative normal equation.

To add in regularization, the equation is the same as our original, except that we add another term inside the ~~par~~ parentheses:

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

where $L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix}$ ← Dimension of matrix $(n+1)\times(n+1)$

L is a matrix with 0 at the top left

and 1's down the diagonal, with 0's everywhere ~~else~~ else. it should have dimension $(n+1) \times (n+1)$, Intuitively, this is the identity matrix (though we are not including x_0), multiple with a ~~sum~~ single real number λ

Example of L dimension:

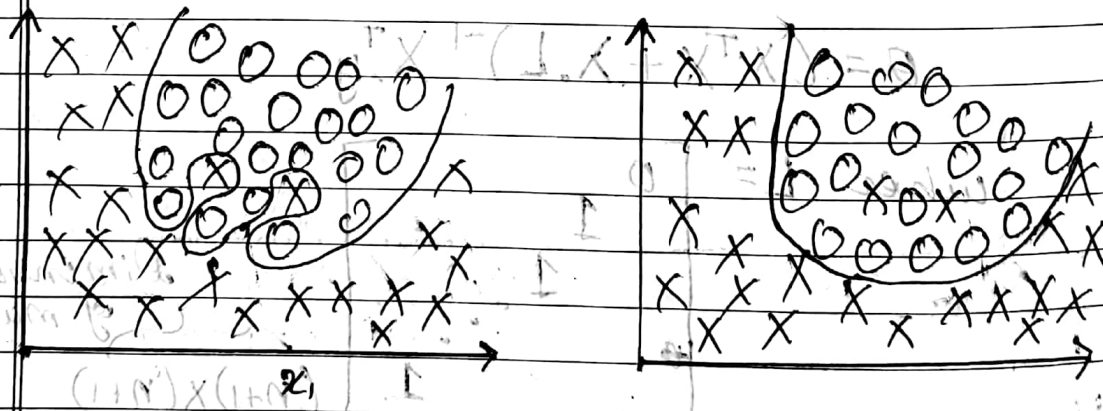
Let $n=2$

then $L = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ ← dimension of matrix $(n+1) \times (n+1)$

Recall that if $m < n$, then $X^T X$ is non-invertible. However, when we add the term $\lambda \cdot L$, then $(X^T X + \lambda \cdot L)$ becomes invertible.

☆ Regularized Logistic Regression

We can regularize logistic regression in similar way that we regularize liner regression.



→ non-Regularized function, over-fit.

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

→ Regularized function → Just Right

Cost function:

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

Regularize this equation by adding.

$$J(\theta) = \frac{-1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2 \quad (j \in \{1,2,3 \ldots n\})$$

The second sum, $\sum_{j=1}^{n} \theta_j^2$ means to explicity exclude the bias term, θ_0.

→ Gradient descent

Repeat {

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \ldots n)$$

}

algorithm is identical but definition is different ('partial derivation of Regularized ~~cost~~ linear Regression cost function and partial derivation of Regularized logistic regression cost function. is totally different)

for regularized cost function on("fminunc(C)")

*enlivo
Date: __/__/__
Page:

★ Advanced optimization. (Octave)

function [jval, gradient] = costfunction (theta)

→ jval = [code to compute $J(\theta)$];

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

→ gradient(1) = [code to compute $\frac{\partial}{\partial \theta_0} J(\theta)$];

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

→ gradient(j) = [code to compute $\frac{\partial}{\partial \theta_{j-1}} J(\theta)$];

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

$j \in \{2, 3 \ldots n+1\}$.

In above function gradient(1) will compute $\frac{\partial}{\partial \theta_0} J(\theta)$, and therfore, in gradient(j) ~~$j \in \{2,3,4 \ldots n+1\}$~~ and $j \in \{2, 3, 4 \ldots n+1\}$

$$\theta = \begin{bmatrix} \theta_0 \leftarrow \text{gradient(1)} \\ \theta_1 \leftarrow \text{gradient(2)} \\ \vdots \\ \theta_n \leftarrow \text{gradient(n+1)} \end{bmatrix}$$

Now, above Costfunction can be parameterize in "fminunc(C)".

☆ Neural Networks :

→ Neural networks

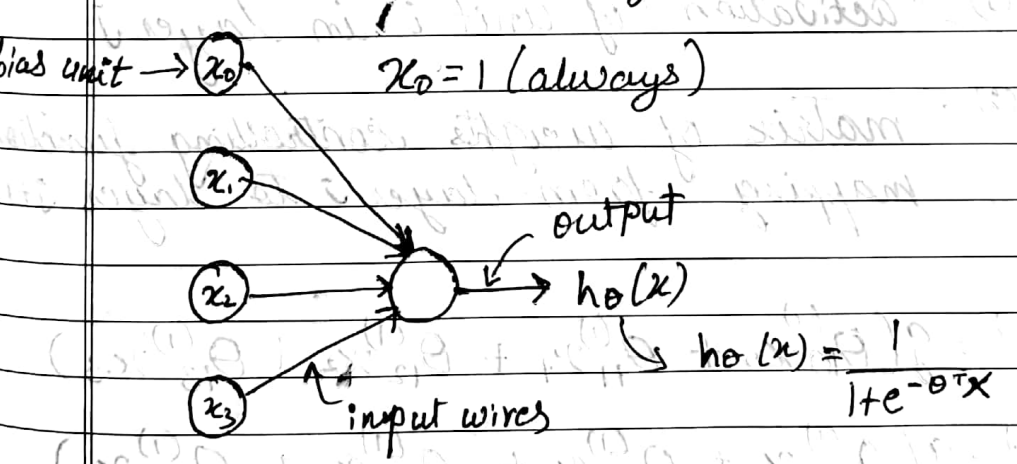
→ Origins : Algorithms that try to mimic the brain.

→ was very widely used in 80s and early 90s ; popularity diminished in late 90s.

→ Recent resurgence : State-of-the-art technique for many application.

☆ Model Representation

→ neuron model : Logistic unit.



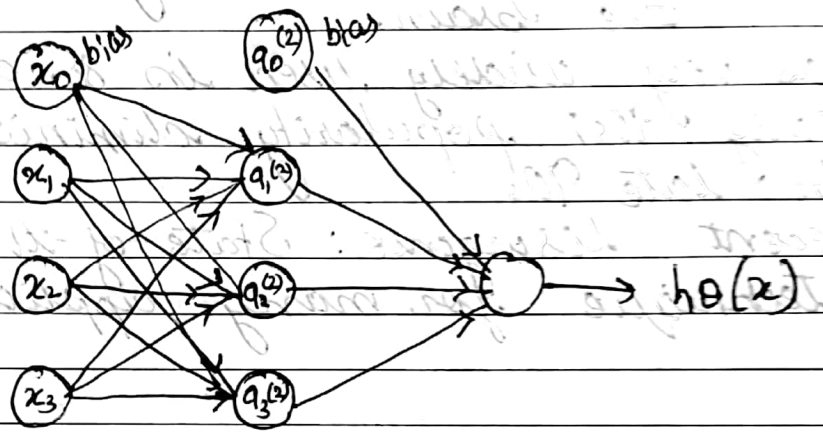
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

θ ↑ weights (parameters)

Sigmoid (logistic) activation function.

→ $g(z) = \frac{1}{1+e^{-z}}$

→ Neural Network



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

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

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

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

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

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

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

→ if network has S_j units in layer j, S_{j+1} units in layer $j+1$, then $\theta^{(j)}$ will be of dimension $S_{j+1} \times (S_j + 1)$.

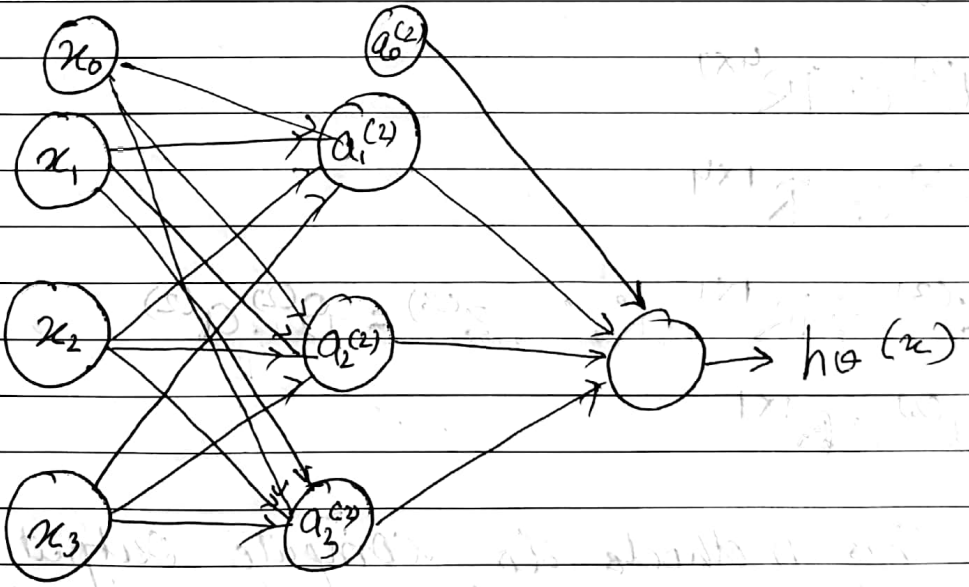
→ it means that dimension of $\theta^{(1)}$ will be "No. of unit in next layer2(excluding bias unit)" X "No. of unit in layer1(including bias unit)"

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

$\theta^{(2)} \in \mathbb{R}^{1 \times 4}$

Example : if layer 1 has 2 input and layer 2 has 4 activation nodes. Dimension of $\theta^{(1)}$ is going to be 4×3 where $S_j = 2$ and $S_{j+1} = 4$, so, $S_{j+1} \times (S_j + 1) = 4 \times 3$.

#



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

$z_2^{(2)} = \theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3$

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

*enlivo
Date: __/__/____
Page: _____

$a_1^{(2)} = g(z_1^{(2)})$

$a_2^{(2)} = g(z_2^{(2)})$

$a_3^{(2)} = g(z_3^{(2)})$

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

$x \in \mathbb{R}^{4\times 1}$

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

$\therefore z^{(2)} \in \mathbb{R}^{3\times 1}$

$z^{(2)} = \theta^{(1)} \cdot x$

$\therefore a^{(2)} \in \mathbb{R}^{3\times 1}$

add $a_0^{(2)} = 1$

$\therefore a^{(2)} \in \mathbb{R}^{4\times 1}$

$\theta^{(2)} \in \mathbb{R}^{1\times 4}$

$\therefore z^{(3)} \in \mathbb{R}^{1\times 1}$ $\because z^{(3)} = \theta^{(2)} \cdot a^{(2)}$

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

$z^{(2)} = \theta^{(1)} \cdot x$

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

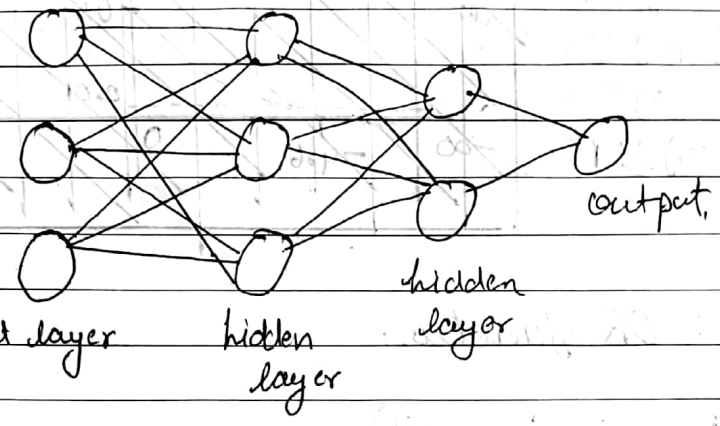
Add $a_0^{(2)} = 1$.

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

$h_\theta(x) = a^{(3)} = g(z^{(3)})$

This methods to compute output layer from input layer thru hidden layers is is called forward propogation.

→ Other network architectures.



★ Non-liner classification example : XOR / XNOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$y = x_1$ XOR x_2

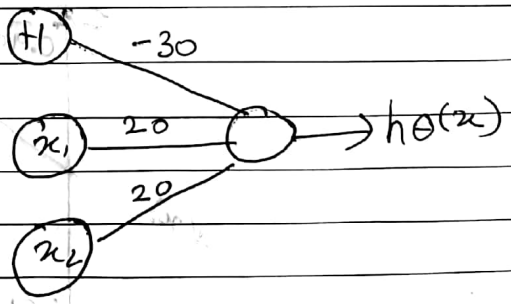
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

$y = x_1$ XNOR x_2
$= \text{NOT}(x_1 \text{ XOR } x_2)$

Start with simple example : AND

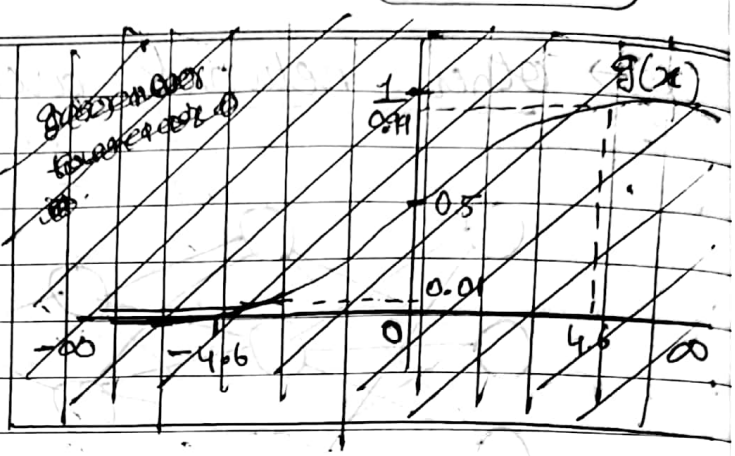
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

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



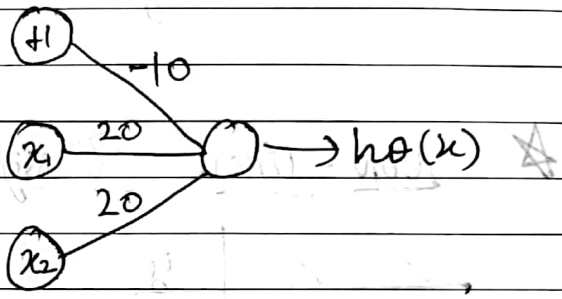
$h\theta(x) = g(-30 + 20x_1 + 20x_2)$
($-30 = \theta_0$, $20 = \theta_1$, $20 = \theta_2$)

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	$g(10) \approx 1$



Example : OR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

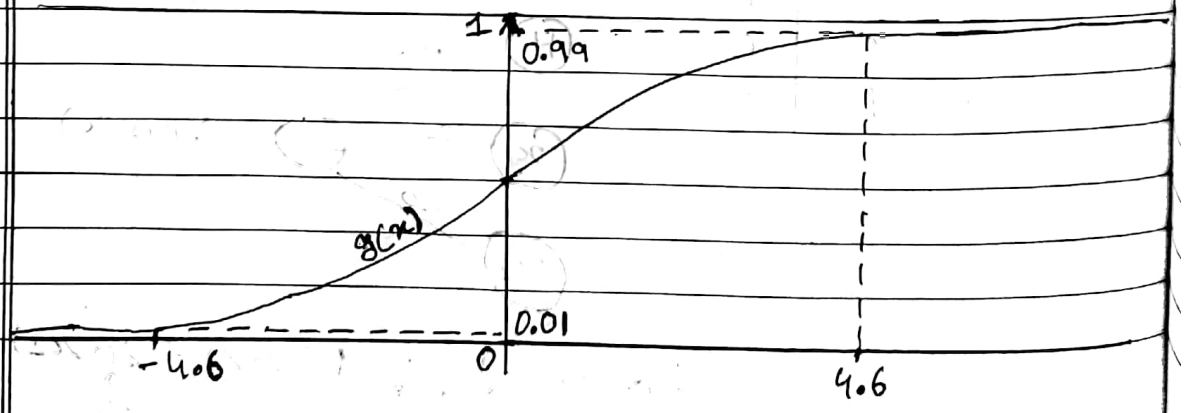


$$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$$

θ_0 θ_1 θ_2

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

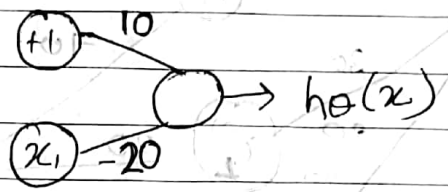
Sigmoid function



*enlivo
Date: ___/___/_____
Page: _____

Example : Not function

x_1	y
0	1
1	0



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

θ_0 θ_1

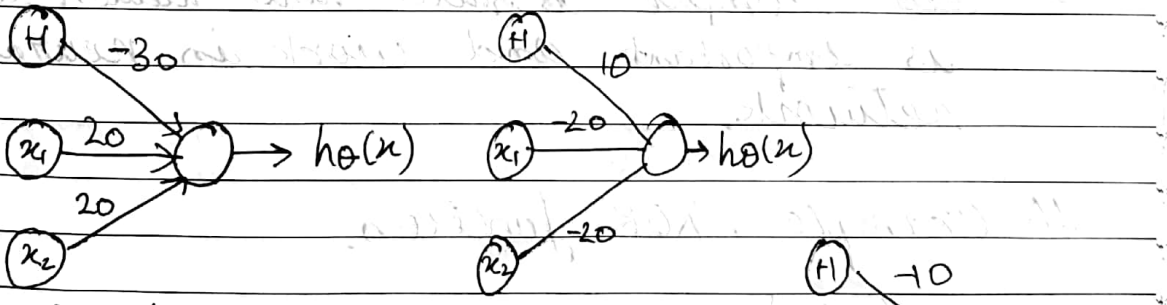
x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

Example : XNOR function

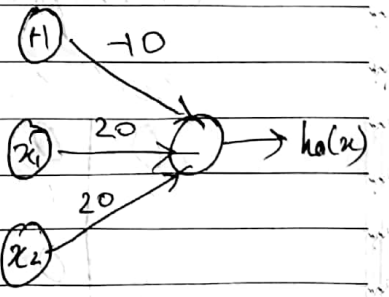
→ ~~Putting it together~~

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

putting it together.



→ x_1 AND x_2



→ (NOT x_1) AND (NOT x_2)

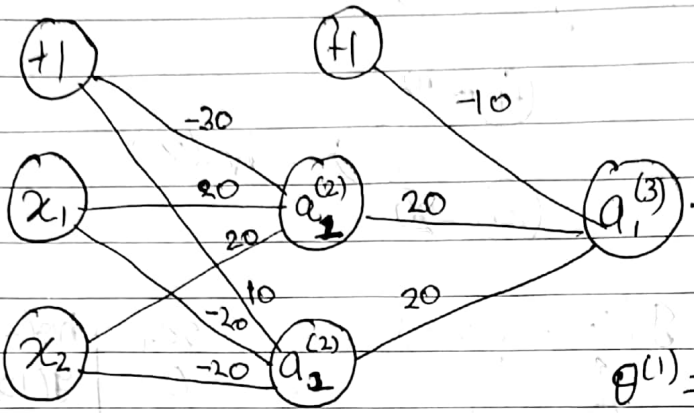
⇒ NOT x_1 = NOT x_2 = 1

if and only if

$x_1 = x_2 = 0$, or

vice-versa.

→ x_1 OR x_2



$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$$

$$\Theta^{(2)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$$

As

$a_1^{(2)} = x_1 \text{ AND } x_2$

$a_2^{(2)} = (\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$

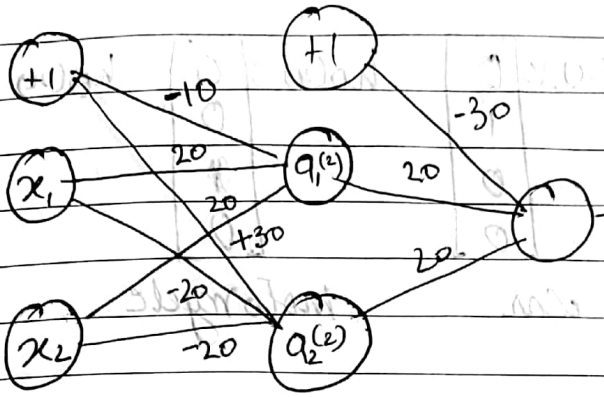
$h_\Theta(x) = a_1^{(2)} \text{ OR } a_2^{(2)}$

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	$g(10) \approx 1$
0	1	0	0	$g(-10) \approx 0$
1	0	0	0	$g(-10) \approx 0$
1	1	1	0	$g(10) \approx 1$

→ this and next example show how hidden layer is important and work in neural network.

Example : XOR function.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



$$\Theta^{(1)} = \begin{bmatrix} -10 & 20 & 20 \\ +30 & -20 & -20 \end{bmatrix}$$

$$\Theta^{(2)} = \begin{bmatrix} -30 & 20 & 20 \end{bmatrix}$$

$a_1^{(2)} = x_1$ OR x_2

$a_2^{(2)} = (\text{NOT } x_1)$ OR $(\text{NOT } x_2)$

$h_\Theta(x) = a_1^{(2)}$ AND $a_2^{(2)}$

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	$g(-10) \approx 0$
0	1	1	1	$g(10) \approx 1$
1	0	1	1	$g(10) \approx 1$
1	1	1	0	$g(-10) \approx 0$

☆ Multiple output unit: one vs all

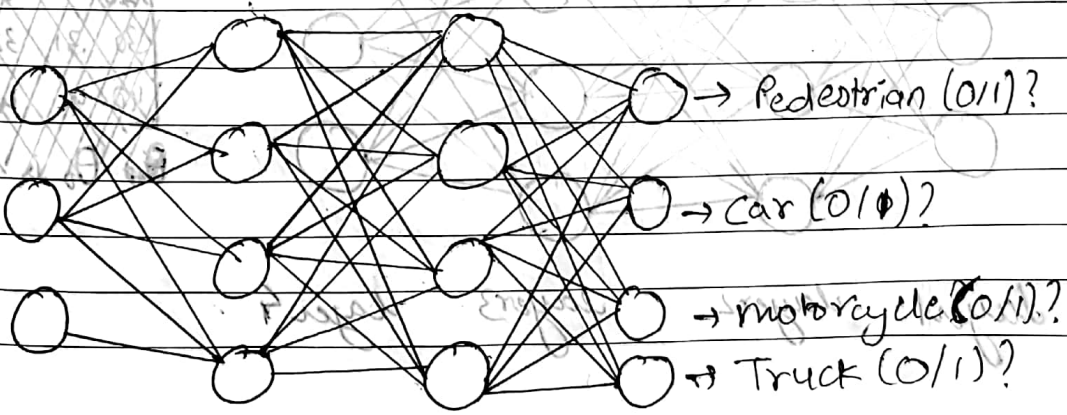
class 1: Pedestrian

class 2: car

class 3: motorcycle

class 4: Truck

$h_\Theta(x) \in \mathbb{R}^{4\times 1}$,



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

when pedestrian car motorcycle truck

Training Set :

$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}) \ldots (x^{(m)}, y^{(m)})$

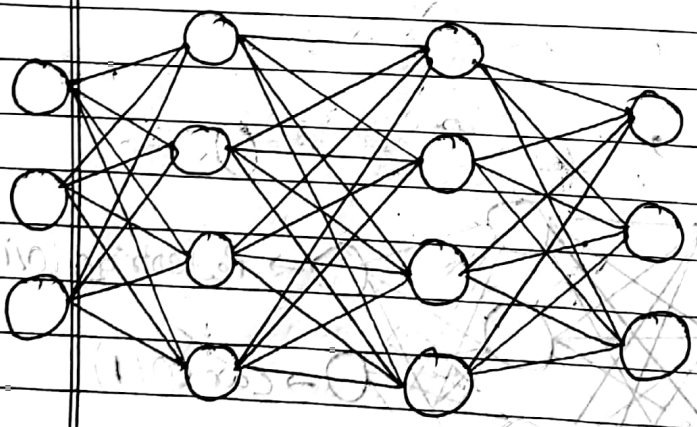
where

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

if pedestrian if car if motorcycle if truck

★ Cost function and Backpropagation

⇒ Neural Network (classification)



layer1 layer2 layer3 layer4

*enlivo
Date : ____/____/_____
Page: _____

Training Examples:

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

L = total no. of ~~layers~~ layers in network
(In this case $L=4$)

S_l = no. of units (not counting bias unit) in layer l
(i.e $S_1=3$, $S_2=4$, $S_3=4$, $S_4=3$)

Binary Classification	Multi-class classification
$(K=1)$ K = number of output units	$(K \geq 3)$
$y = 0$ or 1	$y \in \mathbb{R}^K$
$h_\theta(x) \in \mathbb{R}^{1\times 1}$	E.g: $\begin{bmatrix}1\\0\\0\end{bmatrix}_{3\times 1}$ $\begin{bmatrix}0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\end{bmatrix}$
	$h_\theta(x) \in \mathbb{R}^K$

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:

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

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

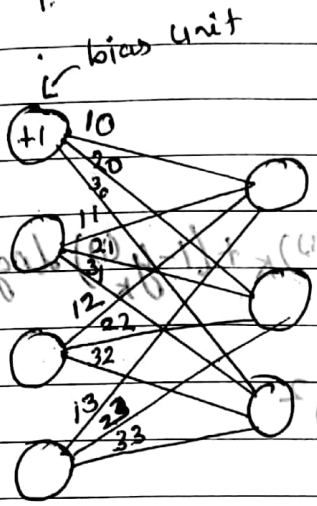
#Q Suppon we want to try to minimize $J(\Theta)$ as a function of Θ, using one of advanced optimization methods. What we need to supply code to compute (as a function of Θ)?

A → We need to supply $J(\Theta)$ and partial derivative term $\frac{\partial}{\partial \Theta_{ij}^{(l)}}$ for every i, j, l

⇒ We have added a few nested summation to account for our multiple output nodes. In the first part of equation, before the square brackets. We have an additional nested summation that loops through the number of output nodes.

In the regularization part, after the square brackets, we must account for multiple theta matrices. The number of ~~cols~~ columns in our current theta matrix is equal to the number of nodes in our current layer (we can include the bias unit but in regularization we exclude bias theta unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit). As before with logistic regression, we square every term.

Example 1:



Theta matrix

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

10	11	12	13
20	21	22	23
30	31	32	33

we will not include Θ values from bias unit in regularization, like we did in regularized logistic regression cost function.

Note:

- the double sum simply adds up the logistic regression costs calculated for each cell in output layer.
- the triple sum simply adds up the squares of all the individual Θs in the entire network. (if we use Θ values from bias unit, it would work about the same and doesn't make big different. But may be this convention of not regularizing the bias term is just slighty more common)
- the i in the triple sum does not refer to training example i.

*enlivo
Date: __/__/____
Page: _____

⇒ ~~Back propagation algorithm~~

Gradient computation.

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

→ $\min_\Theta J(\Theta)$

→ need code to compute

↳ $J(\Theta)$

↳ $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ $\quad \Theta_{ij}^{(l)} \in R^{S_{l+1} \times (S_l+1)}$

In order to compute $J(\Theta)$ we will use above equation and to compute $\frac{\partial}{\partial \Theta_{ij}} J(\Theta)$:

Given one training example (x, y)

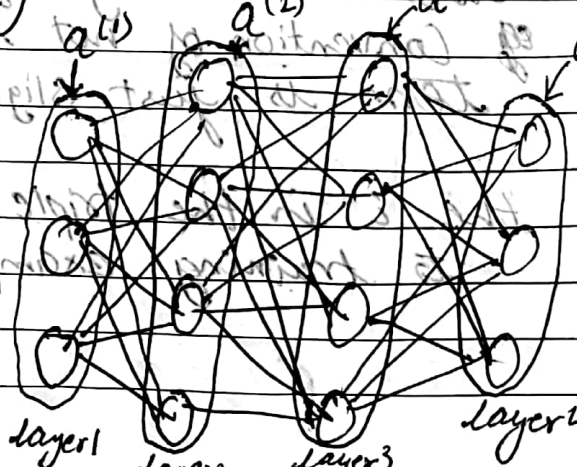
Forward propagation:

$a^{(1)} = x$

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

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

(add $a_0^{(2)}$)



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

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

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

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

⇒ Gradient Computation : Backpropagation algorithm

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

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

⇒ $\delta_j^{(4)} = \underbrace{a_j^{(4)}}_{(h_\theta(x))_j} - y_j$

⇒ Vectorized form :

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

$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$

$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$

.* is element-wise multiplication.

$g'(z^{(i)}) = a^{(i)} .* (1 - a^{(i)})$

for brief $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta^{(l+1)}$ (ignoring λ)

*enlivo
Date: / /
Page:

Backpropagation algorithm:

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

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j).

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

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

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

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

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = \frac{\partial J(\theta)}{\partial a_i^{(l)}} \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial \theta_{ij}^{(l)}}$$

$\delta_i^{(l)}$

← basic idea.

*enlivo
Date: ___/___/___
Page: ___

The capital-delta matrix Δ is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$

Backpropagation ~~Instit~~ Intuition.

Recall that cost function for a neural network is:

$$J(\theta) = -\frac{1}{m} \sum_{t=1}^{m} \sum_{k=1}^{k} \left[y_k^{(t)} \log (h_\theta(x^{(t)}))_k + (1 - y_k^{(t)}) \log (1 - h_\theta(x^{(t)})) \right]$$

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

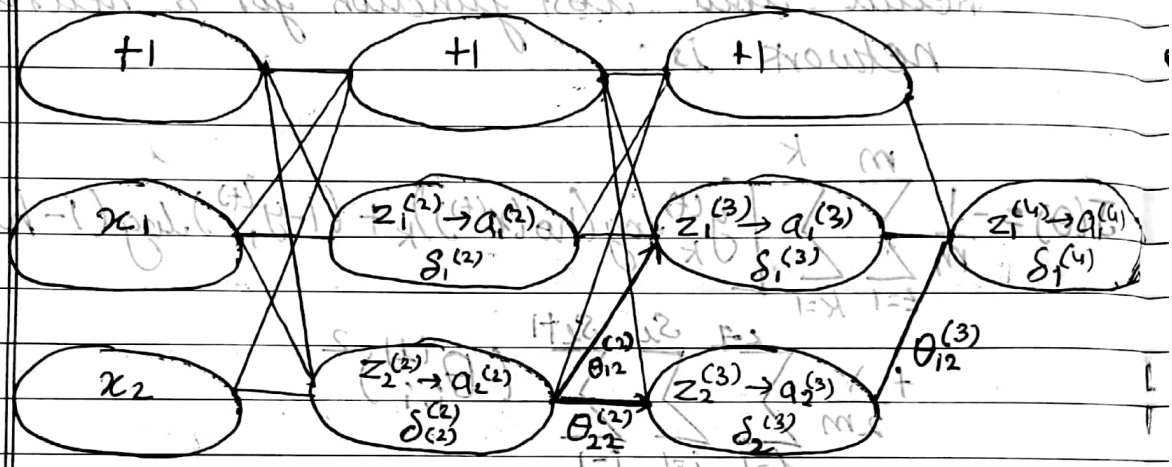
If we consider simple non-multiclass classification ($k=1$) and disregard regularization, the cost is computed with:

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

Intuitively, $\delta_j^{(l)}$ is the "error" for $a_j^{(l)}$ (unit j in layer l). More formally, the delta values are actually the derivative of the cost function.

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(t)$$

Recall that our derivative is the slope of a line tangent to the cost function, so the steeper the slope the more incorrect we are. Let us consider the following neural network below and see how we could calculate some $\delta_j^{(l)}$:



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

formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$ (for $j \geq 0$) where

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

in the ~~image~~, to calculate $\delta_{(2)}^{(2)}$, we multiply the weights $\theta_{12}^{(2)}$ and $\theta_{22}^{(2)}$ by their respective δ value found to the right of each edge. So we get $\delta_2^{(2)} = \theta_{12}^{(2)} * \delta_1^{(3)} + \theta_{22}^{(2)} * \delta_2^{(3)}$. To calculate every single possible $\delta_j^{(l)}$, we could start from the right of our diagram. We

can think of our edges ~~as~~ as our Θ_{ij}. Going from right to left, to calculate the value of $\delta_j^{(l)}$, you can just take the over all sum of each weight times the δ it is coming from. Hence, another example would be

$$\delta_2^{(3)} = \Theta_{12}^{(3)} * \delta_1^{(4)}.$$

octave/matlab

☆ Implementation note: unrolling parameter.

Advanced optimization: (Theta parameter is Vector)

function [jVal, gradient] = CostFunction(theta)
↳ $\mathbb{R}^{n+1}$ ... $\mathbb{R}^{n+1}$

...

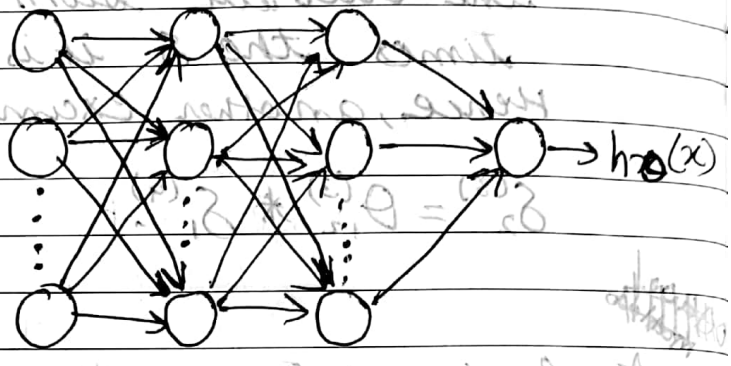
optTheta = fminunc(@costFunction, initialTheta, options)

But with neural networks, we working with sets of matrices:

$\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)
$D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D1, D2, D3)

"Unroll" into Vectors

Example: $S_1 = 10, S_2 = 10, S_3 = 10, S_4 = 1$



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

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

unroll $\Theta^{(1)}, D^{(1)}$ to vector

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];

DVec = [D1(:); D2(:); D3(:)];

# we can get back our original matrices

Theta1 = reshape(thetaVector(1:110), 10, 11)
Theta2 = reshape(thetaVector(111:220), 10, 11)
Theta3 = reshape(thetaVector(221:231), 1, 11)
```

To Summarize:

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

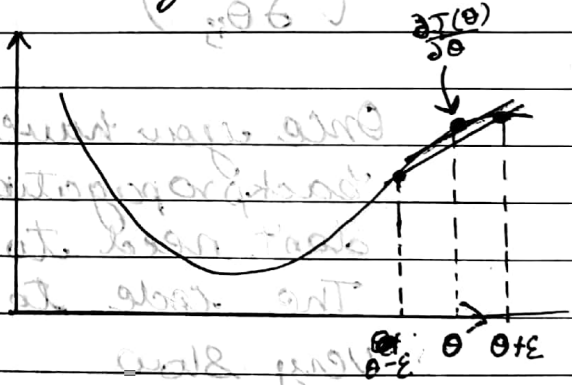
→ unroll to get "initialTheta" to pass to
↳ fminunc (@costfunction, initialTheta, options)

→ function [jVal, gradientVec.] = costfunction (thetaVec)
↳ From thetaVec., get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
↳ use forward/back propogation to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$.
↳ Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get "gradientVec".

☆ Gradient checking (Numerical estimate of gradient)

Gradient checking will assure that our backpropagation works as intended. we can approximate the derivative of our cost function with:

$$\frac{\partial}{\partial\Theta} J(\Theta) \approx \frac{J(\Theta+\epsilon) - J(\Theta-\epsilon)}{2\epsilon}$$



With multiple theta matrices, we can approximate the derivative with respect to Θ_j as follows:

$$\frac{\partial}{\partial\Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \ldots, \Theta_j+\epsilon, \ldots, \Theta_n) - J(\Theta_1, \ldots, \Theta_j-\epsilon, \ldots, \Theta_n)}{2\epsilon}$$

A small value for ϵ (ϵ positive), such as $\epsilon = 10^{-4}$, guarantees that math work out properly.

we are only adding or substracting epsilon to the θ_i matrix. In octave we can do it as follows:

```
epsilon = 1e-4;
for i = 1:n,
    thetaplus = theta;
    thetaplus(i) += epsilon;
    thetaMinus = theta;
    thetaMinus -= epsilon;
    gradApprox(i) = (J(thetaplus) - J(thetaminus))
                    / (2*epsilon)
end;
```

So once we compute our "gradApprox" vector, we can check that gradApprox $\approx$ deltaVector $\left(\frac{\partial J(\theta)}{\partial \theta_i}\right)$

Once you have verified once that your backpropagation algorithm is correct, you don't need to compute gradApprox again. The code to compute gradApprox can be very slow

important:

Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of costFunction(...)) your code will be very slow

☆ Random Initialization ☆

Initializing all theta weights to zero does not work with neural networks. When we backpropagate, all nodes will update to the same value repeatedly. Instead we can randomly initialize our weights for our Θ matrices using the following methods:

Random initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$ (i.e $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)
and
dimension of $\Theta_{ij}^{(1)} \in \mathbb{R}^{10 \times 11}$, $\Theta_{ij}^{(2)} \in \mathbb{R}^{1 \times 11}$
(i.e Theta1 is 10x11 dimension and Theta2 is 1x11 dimension)

E.g

```
Theta1 = rand(10,11) * (2*INIT_EPSILON) -
                INIT_EPSILON;
Theta2 = rand(1,11) * (2* INIT_EPSILON) -
                INIT_EPSILON;
```

We initialize each $\Theta_{ij}^{(l)}$ to a random value between $[-\epsilon, \epsilon]$. Using the above formula guarantees that we get the desired bound. The same procedure applies to all the Θ's.

→ rand(x,y) is just function in Octave that will initialize a matrix of random real numbers between 0 & 1.

→ Note: The Epsilon used above is unrelated to the epsilon from Gradient checking.

☆ Putting it Together

→ First, pick a network architecture; choose the layout of your neural network, including how many hidden units in each layer and how many layers in total you want to have.

- Number of input units = dimension of feature $x^{(i)}$
- Number of output units = number of classes.
- Number of hidden units per layer = usually more the better (must balance with cost of computation as it increase with more hidden units)
- Default: 1 hidden layer. if you have more than 1 hidden layer, then it is recommended that you have the same number of units in every hidden layer.

Training a neural network

1. Randomly initialize the weights
2. Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$
3. Implement the cost function
4. Implement back propagation to compute partial derivatives

For $i = 1:m$

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

(Get activation $a^{(l)}$ and delta term $\delta^{(l)}$ for $l = 2 \dots L$)

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

end;

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

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

→ Then disable gradient checking code.

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

⇒ $J(\Theta)$ - will be non-convex.

*enlivo
Date: ___/___/_____
Page: _____

★ Debugging a learning algorithm.

Suppose you have implemented regularized linear regression to predict housing prices.

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

However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- Get more training example.
- Try smaller sets of features
- Try getting additional features ←
- Try adding polynomial features e.g. $(x_1^2, x_2^2, x_1x_2, \text{etc})$.
- Try decreasing λ
- Try increasing λ

Diagnostic

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

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

*enlivo
Date: __/__/____
Page: _____

Evaluating a hypothesis

We can move on to evaluate our new hypothesis.

A hypothesis may have a low error for the training Examples but still be inaccurate (because of overfitting)
Thus, to evaluate a hypothesis, given a dataset of training example, we can split up the data into two sets: a training set and a test set

The new procedure using these two sets is then:

1. learn Θ and minimize $J_{train}(\Theta)$ using the training set.
2. compute the test set error $J_{test}(\Theta)$

The test set error

1) For linear regression: $J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2$

2) For classification ~ Misclassification error (~~of misclassif~~ (a.k.a 0/1 misclassification error):

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

This gives us a binary 0 or 1 error result based on a misclassification. The average test error for the test set is:

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

This gives us the proportion of the test data that was misclassified.

☆ Model Selection and Train / Validation / Test Sets

Just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis. It could overfit and as a result your prediction on the test set would be poor. The error of your hypothesis as measured on the data set with which you trained the parameter will be lower than the error on any other data set.

Given many models with different polynomial degrees we can use a systematic approach to identify the 'best' function. In order to

choose the model of your hypothesis, to you can test each degree of polynomial and look at the error result!

One way to break down our dataset into the three sets is:

- Training set: 60%
- Cross Validation Set: 20%
- Test set: 20%

We can now calculate three separate error values for the to three different sets using the following methods:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the test least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d= theta from polynomial with lowes error);

This way, the degree of polynomial d has has not been trained using test set.

★ Diagnosing Bias Vs. Variance

In this topic we examine the relationship between the degree of the polynomial d and the underfitting or overfitting of our hypothesis.

- We need to distinguish whether bias or variance is the problem contributing to bad prediction.
- High bias is underfitting and high variance is overfitting, ideally, we need find a golden mean between these two.

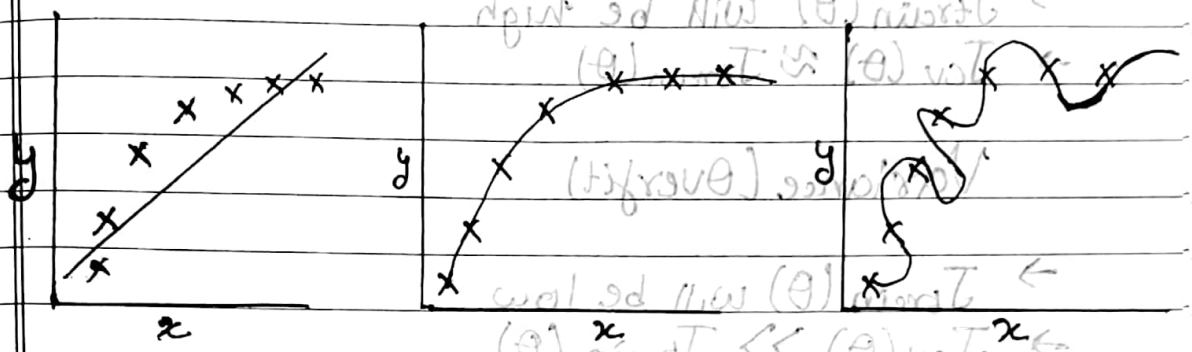
→ Training error will tend to decrease as we increase the degree d of the polynomial.

At the same time, the cross validation error will tend to decrease as we increase d up to a point, and then it will increase as d is increased forming a convex curve.

High bias (underfitting): both $J_{train}(\theta)$ and $J_{cv}(\theta)$ will be high. Also, $J_{cv}(\theta) \approx J_{train}(\theta)$

High Variance (Overfitting): $J_{train}(\theta)$ will be low and $J_{cv}(\theta)$ will be much greather than $J_{train}(\theta)$.

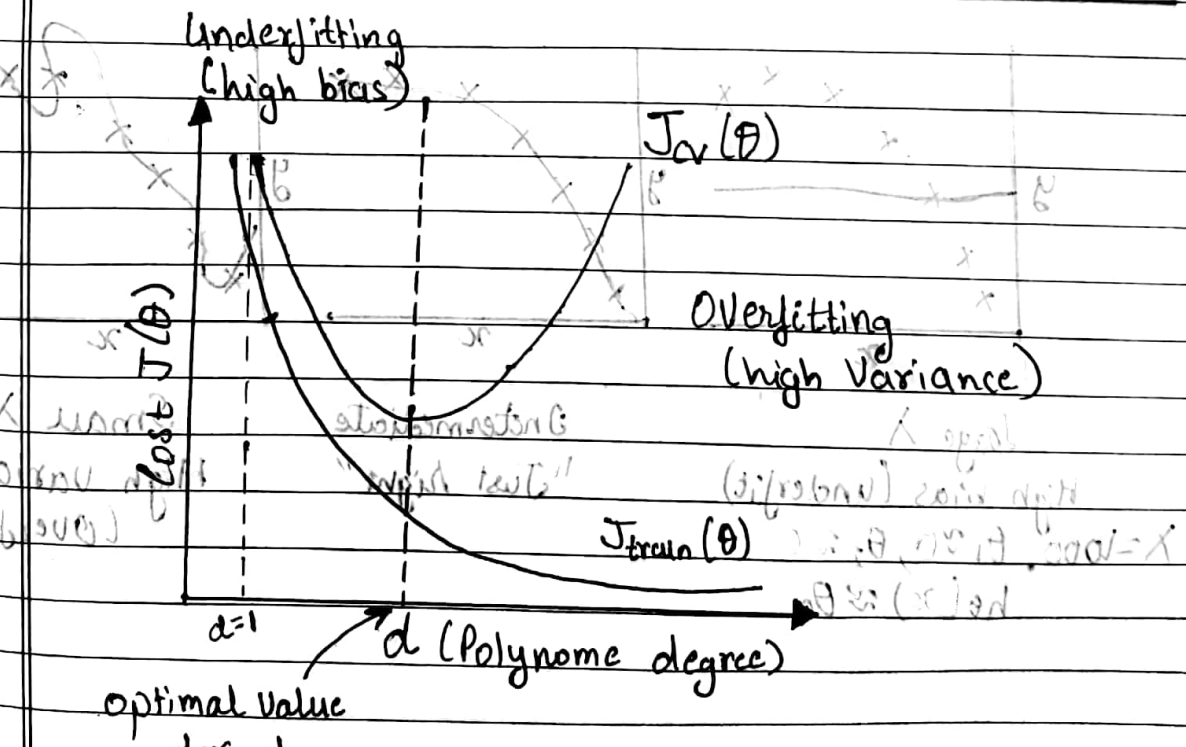
- Bias/variance.



$\theta_0 + \theta_1 x$	$\theta_0 + \theta_1 x + \theta_2 x^2$	$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$
$(d=1)$	$(d=2)$	$(d=4)$ ☆
High bias (underfitting)	Just right	High Variance (overfitting)

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

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



Bias (underfit)

→ $J_{train}(\theta)$ will be high
→ $J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit)

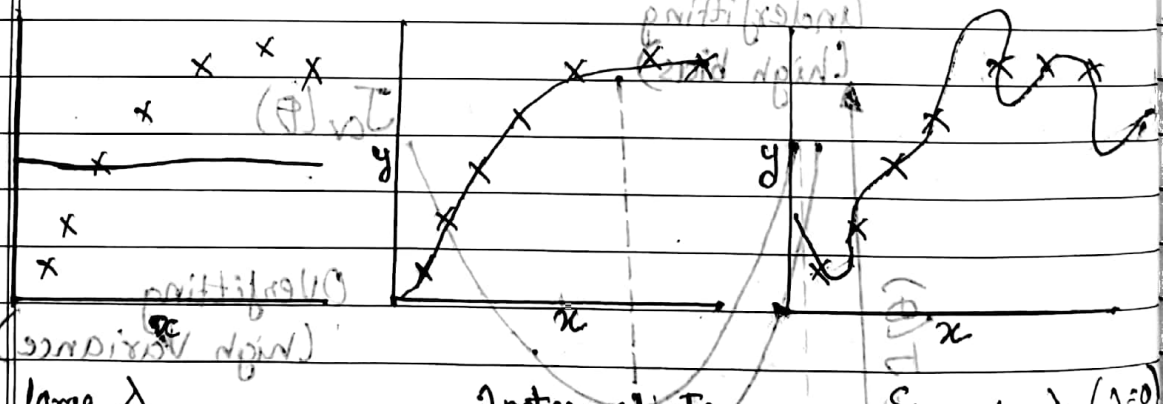
→ $J_{train}(\theta)$ will be low
→ $J_{cv}(\theta) >> J_{train}(\theta)$

☆ Regularization and Bias / Variance

Linear regression with regularization.

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

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



Large λ
High bias (underfit)
$\lambda = 10000$, $\theta_1 \approx 0, \theta_2 \approx 0$
$h_\theta(x) \approx \theta_0$

Intermediate
"Just right"

Small λ ($\lambda = 0$)
High variance
(overfit)

*enlivo
Date: __/__/____
Page: _____

We see that as λ increases, our fit becomes more rigid. On the other hand, as λ approaches 0, we tend to overfit the data, So how do we choose our parameter λ to get it "just right"? In order to choose the model and the regularization term we need to:

1. Create a list of lambdas (i.e $\lambda \in \{0, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56, 5.12, 10.24\}$);
2. create a set model with different degree or any other variants.
3. Iterate through the λs and for each λ go through all the models to learn some θ.
4. Compute the cross validation error using the learned θ (computed with λ) on the $J_{cv}(\theta)$ <u>without regularization</u> or <u>$\lambda = 0$</u>
5. Select the best combo that produces the lowest error on the cross validation set.
6. Using the best combo θ and λ, apply it on $J_{test}(\theta)$ to see if it has a good generalization of the problem.

E.g

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

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

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

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

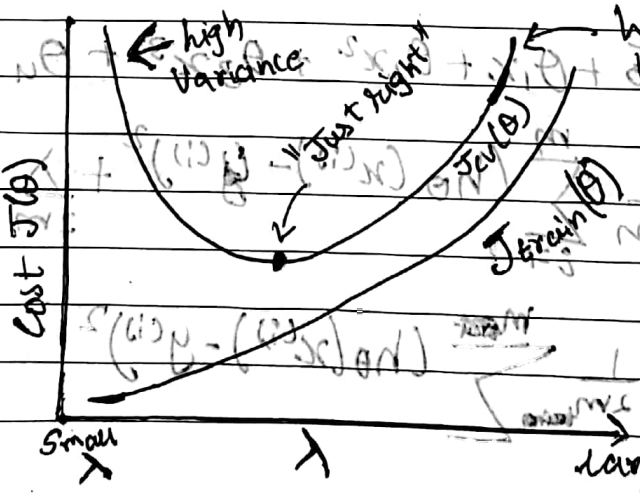
$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)}\right)^2$$

1. try $\lambda = 0 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
2. try $\lambda = 0.01 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
3. try $\lambda = 0.02 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(3)} \rightarrow J_{cv}(\theta^{(3)})$
4. try $\lambda = 0.04 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(4)} \rightarrow J_{cv}(\theta^{(4)})$
5. try $\lambda = 0.08 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(5)} \rightarrow J_{cv}(\theta^{(5)})$
6. ⋮
12. try $\lambda = 10 \rightarrow \min_\theta J(\theta) \rightarrow \theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$

Now, pick θ with lowest $J_{cv}(\theta)$.
Say, $\theta^{(5)}$

Now, ~~find~~ apply it on $J_{test}(\theta^{(5)})$ to see Test Error (to see if has good generalization of the problem)

Bias/Variance as a function of the λ



Learning Curves

Training an algorithm on a very few number of data points (such as 1, 2, or 3) will easily have 0 errors because we can always find a quadratic curve that touches exactly those number of points. Hence:

- As the training set gets larger, the error for a quadratic function increases.
- The error value will plateau out after a certain m, or training set size.

Experiencing high bias:

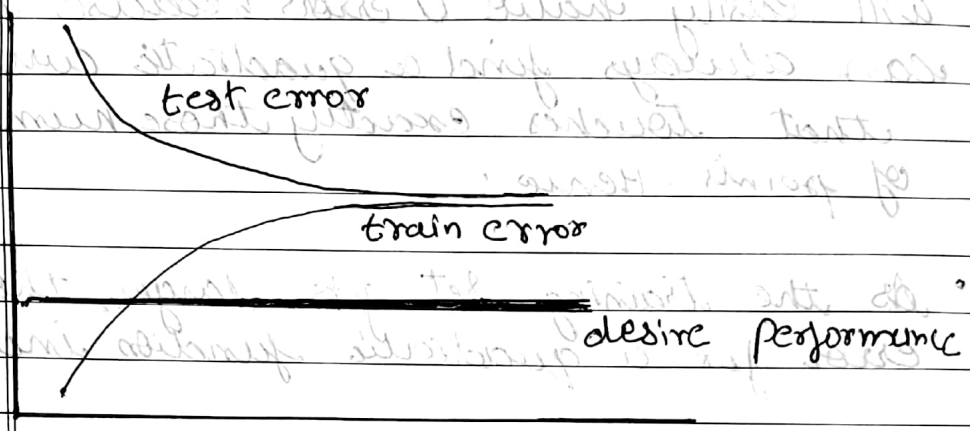
Low training set size: Causes $J_{train}(\theta)$ to be low and $J_{cv}(\theta)$ will be high.

Large training set size: Causes $J_{train}(\theta)$ and $J_{cv}(\theta)$ to be high with $J_{train}(\theta) \approx J_{cv}(\theta)$.

If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

More on Bias Vs. Variance.

Typical learning curve for high bias (at fixed model complexity)



Experiencing high Variance:

Low training set size: $J_{train}(\theta)$ will be low and $J_{cv}(\theta)$ will be high.

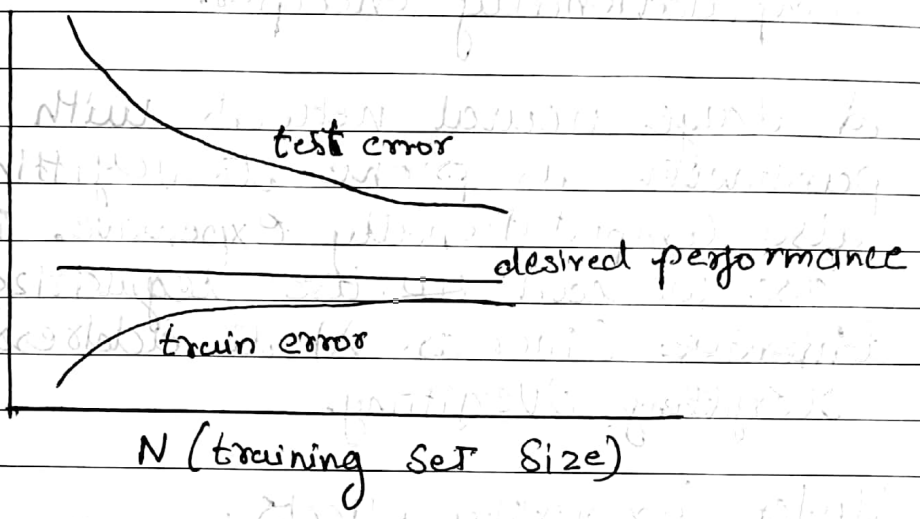
Large training set size: $J_{train}(\theta)$ increases with ~~test~~ training set size and $J_{cv}(\theta)$ continues to decrease without leveling off.

Also, $J_{train}(\theta) < J_{cv}(\theta)$ but the difference between them remains ~~significant~~ Significant.

if a learning algorithm is suffering from high variance, getting more

training data is likely to help.

Typical Learning curve for high Variance (at fixed model complexity)



* Our decision process can be broken down as follows:

- Getting more training examples : fixes high Variance
- ~~Trying~~ Trying smaller sets of ~~feat~~ features : fixes high Variance
- Adding features : fixes high bias
- Adding polynomial features : fixes high bias.
- Decreasing λ : fixes high bias.
- Increasing λ : fixes high Variance.

Diagnosing Neural Networks.

- A neural network with fewer parameters is prone to underfitting. It is also computationally cheaper.
- A large neural network with more parameter is prone to verfitting. it is also computationally expensive. in this case we can ~~use~~ use regularization ~~increase~~ (increase λ) to address the ~~overfitting~~ Overfitting.

Model Complexity Effects:

- Lower-order polynomials (low model complexity) have high bias and low variance. In this case, the model fits poorly consistently.
- Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very ~~to~~ high variance.
- In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

Error analysis

Recommended approach

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

for example:

m_{cv} = 500 examples in cross validation set
algorithm misclassifies 100 emails
manually examine 100 errors, and categorize them based on:
(i) what type of email it is!
(ii) what cues (features) you think would have helped the algorithm classify them correctly.

Error metrics for skewed classes

Cancer Classification example

→ Train logistic regression model $h_\theta(x)$. ($y=1$ if cancer, $y=0$ otherwise)

Find that you got 1% error on test set. (99% correct diagnoses)

→ only 0.50% of patients have cancer

↳ Skewed classes (imbalance data)

→ it means its 99.5% of Patients do not have cancer.

Precision / Recall

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

Predicted class	Actual class: 1	Actual class: 0
1	True positive	False positive
0	False negative	True negative

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

$$\frac{\text{True positive}}{\text{True positive + False positive}} = \frac{\text{True Positive}}{\text{Predicted Positive}}$$

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

$$\frac{\text{True positive}}{\text{True positive + false negative}} = \frac{\text{True positives}}{\text{actual positives.}}$$

⇒ Suppose we want to predict $y=1$ (cancer) only if very confident. : Higher precision, lower recall.

⇒ Suppose we want to avoid missing too many cases of cancer (avoid false negatives): Higher recall, lower precision.

* F, Score (F score)

How to compare precision/recall numbers?

*enlivo
Date: ____/____/____
Page: _____

P = Precision
R = Recall

$$f_1\,score = 2 * \frac{PR}{P+R}$$

$P=0$ or $R=0$ $\Rightarrow$ $f_1\,score = 0$

$P=1$ or $R=1$ $\Rightarrow$ $f_1\,score = 1$

*enlivo
Date: __/__/____
Page: _____

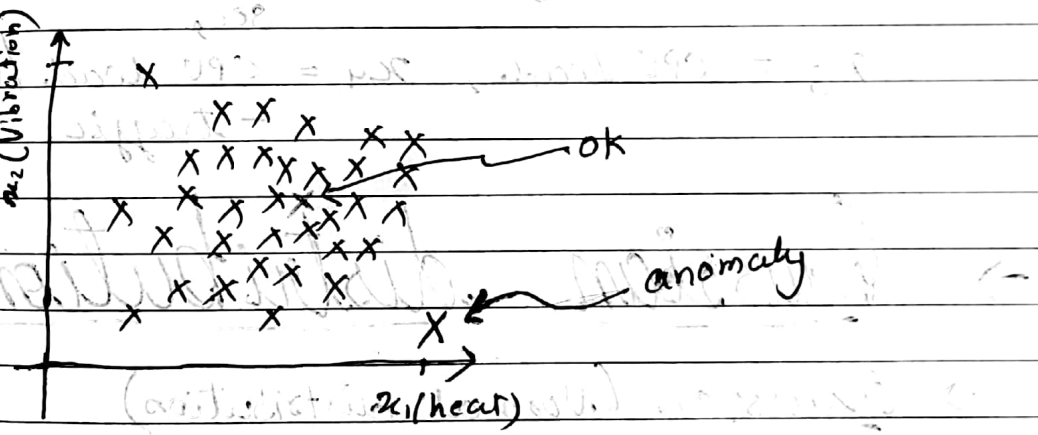
Anomaly detection

→ problem motivation.

Anomaly detection example.
x_1 = heat generated
x_2 = Vibration intensity.
⋮

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

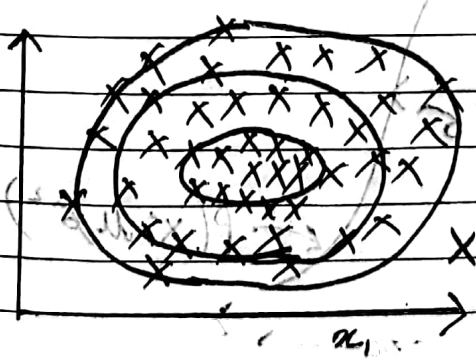
New data : x_{test}



Density estimation.

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

→ Is x_{test} anomalous? model : $P(x)$



$P(x_{test}) < \varepsilon \rightarrow$ flag anomaly

$P(x_{test}) \geq \varepsilon \rightarrow$ OK

Anomaly detection example

→ fraud detection:

→ $x^{(i)}$ = feature of user i's activities.
→ model $P(x)$ from data.
→ Identify unusual users by checking which have $P(x) < \varepsilon$

⇒ Manufacturing

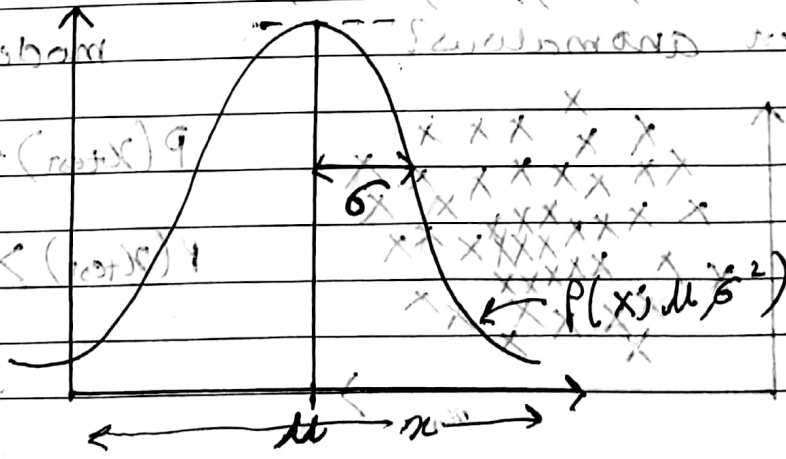
⇒ Monitoring computers in a data center.

$x^{(i)}$ = feature of machine i
x_1 = memory use, x_2 = number of disk accesses/sec,
x_3 = CPU load, x_4 = CPU load / network traffic.

→ Gaussian distribution

→ Gaussian (Normal distribution)

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

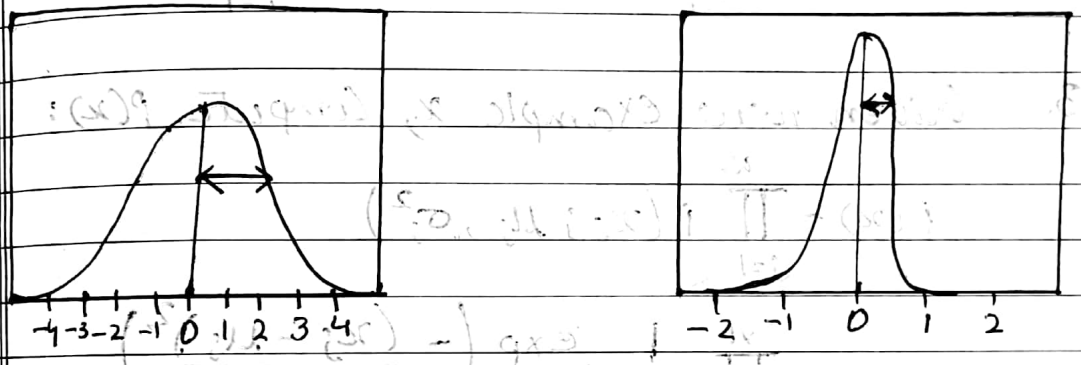


σ : standard deviation, σ^2 : variance, μ : mean

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

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

→ Gaussian distribution example



$\mu = 0$, $\sigma = 2$ $\quad$ $\mu = 0$, $\sigma = 0.5$

→ parameter estimation:

Dataset : $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $\quad x^{(i)} \in \mathbb{R}$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \qquad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

→ Algorithm

training set : $\{x^{(1)}, \ldots, x^{(m)}\}$, Each example is $x \in \mathbb{R}^n$

$$P(x) = P(x_1; \mu_1, \sigma_1^2) \cdot P(x_2; \mu_2, \sigma_2^2) \cdot P(x_3; \mu_3, \sigma_3^2) \cdots P(x_n; \mu_n, \sigma_n^2)$$

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

→ Anomaly detection algorithm:

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

2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

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

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 $p(x) < \varepsilon$

→ The important of real-number evaluation.

When developing a learning algorithm (choosing features, etc.) making decisions is much easier if we have a way of evaluating our learning algorithm)

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

Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (assume normal examples / not anomalous)

$\prod_{i=i}$

- Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
- Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

→ Aircraft engines motivating example.

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

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

→ Algorithm evaluation.

- 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 cross validation 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 common)	→ large number of positive and negative examples.
→ large number of negative ($y=0$) examples.	
→ many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like;	→ Enough Positive example for algorithm to get a sense of what positive example are like, future positive example likely to be similar to ones in training set.
→ future anomalies may look nothing like any of the anomalous examples we've seen so far.	
→ Example:	→ example:
• Manufacturing (e.g. aircraft engines)	• Email Spam classification
• monitoring machine in a data center	• Weather prediction
	• Cancer classification.