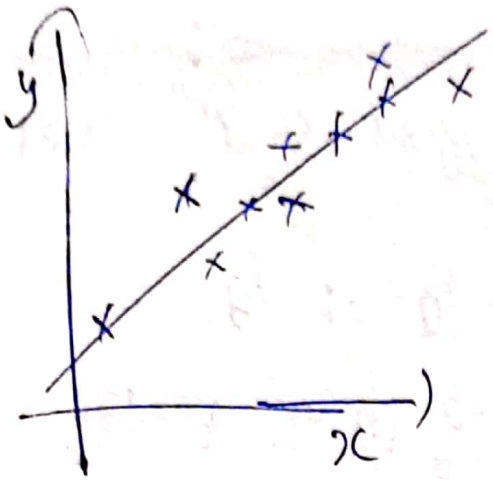


here :

m : size of training set

x : input param

y : output (given)

h : hypothesis function

h is used to predict 'y', given 'x', after learning from m data sets.

h(x) → straight line ⇒ $h(x_i) = \theta_0 + \theta_1 x_i$

~~needs~~

x_i → i^{th} training set input

y_i → i^{th} training set output (given)

$h(x_i)$ → i^{th} training set (predicted)

~~h should be as~~ close

h should have minimal error w.r.t. y.

⇒

let $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x_i) - y_i \right)^2$

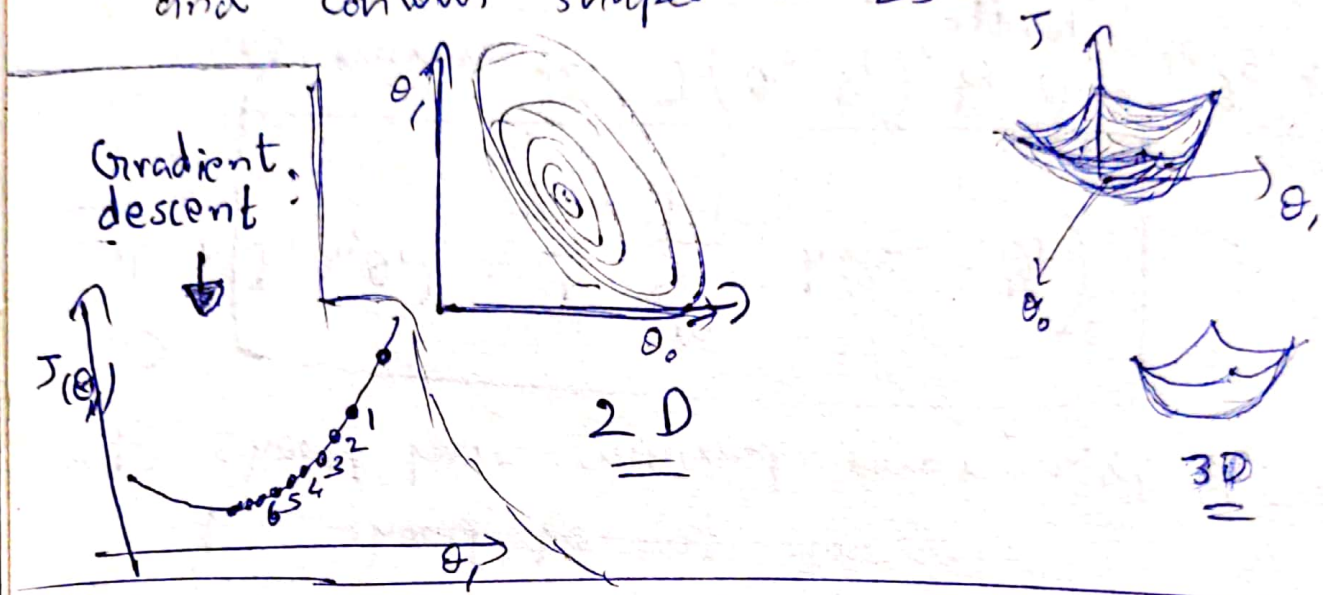
⇒ minimise $J(\theta_0, \theta_1)$, by choosing appropriate θ_0 & θ_1

Q] Why is $\frac{1}{2}$ is there?

Answer] In order to cancel with square term, upon differentiation.

Contour Plots

when $J(\theta_0, \theta_1)$ is plotted as a fn of θ_0 & θ_1 the plot is a 'bowl' shaped in 3D, and 'contours' shaped in 2D--



How to find $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$? :- by using

Gradient Descent

Algo :-

1) start with some θ_0, θ_1
2) Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until minima is reached.

mathematically speaking :-

repeat until Convergence :-

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

here, $\alpha \Rightarrow$ learning rate

$:=$ sign $\Rightarrow$ ~~Continuous~~ simultaneous update.

Simultaneous Update :-

$$temp_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} (J(\theta_0, \theta_1))$$

$$temp_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} (J(\theta_0, \theta_1))$$

$$\theta_0 = temp_0$$

$$\theta_1 = temp_1$$

i.e. both θs should get updated simultaneously ∴ J is a function of two variables (θ_0, θ_1).

Learning Rate (α) :-

(1) If α is too small, gradient descent can be slow.

(2) If α is too large, gradient descent can overshoot the minima. It can fail to converge, or even diverge.

(3) we can keep α fixed, and still reach a local minimum because as we ~~tend~~ converge towards local minimum, gradient descent will automatically take smaller steps.

Now, we know that the gradient descent algorithm is :-

repeat until convergence :-

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

↳ (1)

We also know :-

Linear regression model : $h_\theta(x) = \theta_0 + \theta_1 x$

&

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x_i) - y_i \right)^2$$

~~from~~ 1 ↳ (2)

from (1) & (2) :-

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(h(x_i) - y_i \right)^2 \right)$$

$h(x_i) \rightarrow \theta_0 + \theta_1 x_i$

$$\Rightarrow \theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} \left(\frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x_i - y_i \right)^2 \right)$$

$\Rightarrow$ for ~~On each of~~ upon applying partial differentiation $j = 0$ & $j = 1$ terms, we get :

we get :-

Gradient descent Algo for Linear Regression :-

(in one variable)

repeat until Convergence :-

{

$$\theta_0 := \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h(x_i) - y_i \right)$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} \left(\left(h(x_i) - y_i \right) * x_i \right)$$

}

Batch Gradient Descent :-

Each step uses all training set examples.

ex :- the above algorithm is a batch gradient descent.

Linear Regression in Multiple Variables :-

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots \theta_n x_n$$

where

$n \rightarrow$ no. of features

For uniformity's sake let us introduce a new variable $x_0 = 1$

$$\Rightarrow \boxed{h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots \theta_n x_n}$$

Gradient Descent for linear Regression (Multiple variables)

repeat until convergence:-

{

$$\theta_0 := \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} \left[\left(h(x_i) - y_i \right) x_{0i} \right]$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} \left[\left(h(x_i) - y_i \right) x_{1i} \right]$$

$$\theta_2 := \theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \left[\left(h(x_i) - y_i \right) x_{2i} \right]$$

$\vdots$

θ_n

}

(i) Tips :-

(1) If gradient descent is not working for certain α, decrease it and try again.

For sufficiently small α, J should decrease with every iteration.

(2) α too small $\Rightarrow$ slow convergence

α too large $\Rightarrow$ no convergence (or) even divergence!

(3) You can declare convergence if J decreases by less than ε in one iteration. $\varepsilon \rightarrow$ user defined (ex:- 10^{-3})

Polynomial Regression :-

Tip :- we can combine multiple features into one.

ex :- $x_3 = x_1 . x_2$

Polynomial Regression :- our hypothesis function need not be linear, if it doesn't fit the data well.

~~Ex~~

Linear :- $\theta_0 + \theta_1 x_1$

Quadratic :- $\theta_0 + \theta_1 x_1 + \theta_2 x_1^2$

cubic :- $\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

squareRoot :- $\theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

let $x_2 = x_1^2$ & $x_3 = x_1^3$

$\Rightarrow$ we have created 2 new features in cubic function.

(*) If we square (or) cube a variable, then its range of values also become squared / cubed. In order to make plots easier, we **Scale** the features into **appropriate range**.

↳ upto you :-

$-1 \leq x_i \leq +1$

$-0.5 \leq x_i \leq +0.5$

Feature Scaling & Mean Normalization :

We use these two techniques to reduce the range of each feature.

Why? :- → ~~Faster~~ plots made easier
→ Fastly arrive at minima.

$$x_i := \frac{x_i - \mu_i}{\sigma_i}$$

mean normalization

$x_i - \mu_i$ → mean

Feature Scaling

dividing by σ_i → standard deviation.

If calculating σ_i is difficult,
~~get~~ replace it by range of that x_i
(max. value of x_i − min value of x_i)

Classification :-

Output values are discrete and finite. (and small number)

Linear Regression doesn't work well in classification problems, since classification is not a linear function.

Binary classification :- output takes 2 values (0 (or) 1).

In order to make Linear Regression work for Binary classification, we use a different form of hypothesis function, called :-
Logistic (or) Sigmoid Function.

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

we can plug $\theta^T x$ into z, to get our new hypothesis function :-

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

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

where θ^T → Transpose of vector :- $\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

i.e. $\theta^T x = \theta_0 x_0 + \theta_1 x_1 + \cdots \theta_n x_n$

we can also say that :-

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

read as :- Probability that $y=1$, given x, parameterized by θ.

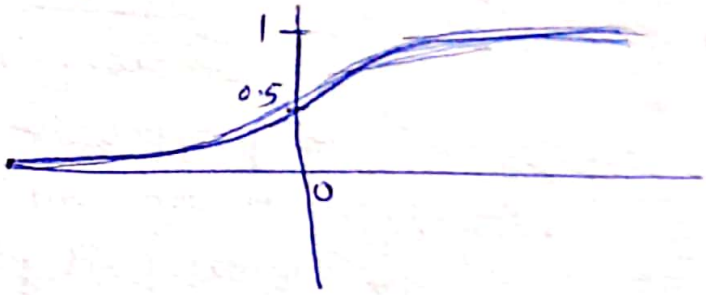
meaning :-

$h_\theta(x)$ will give probability that o/p is 1.

ex :- if $h_\theta(x) = 0.7$ ⇒ Probability of $y=1$ is 0.7 (70%)

Logistic function (or) Sigmoid function & its behaviour :-

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



From the figure, we can say that :-

- ~~$g(z) \geq 0.5 \Rightarrow y=1$~~
- $g(z) < 0.5 \Rightarrow y=0$

$g(z) \geq 0.5$ when $z \geq 0$ → (1)

also, when :-

$z = 0$, $g(z) = \frac{1}{2}$

$z = +\infty$, $g(z) = 1$

$z = -\infty$, $g(z) = 0$

In order to fit this function into our Binary classification, we can translate our o/p of hypothesis function :-

$$\text{If } h_\theta(x) \geq 0.5 \Rightarrow y = 1$$
$$\text{If } h_\theta(x) < 0.5 \Rightarrow y = 0 \quad \rightarrow (2)$$

now, replacing z with $\theta^T x$ in (1), we get!

$$g(\theta^T x) \geq 0.5 \quad \text{when} \quad \theta^T x \geq 0 \quad \rightarrow (3)$$

from (2) & (3), we get!-

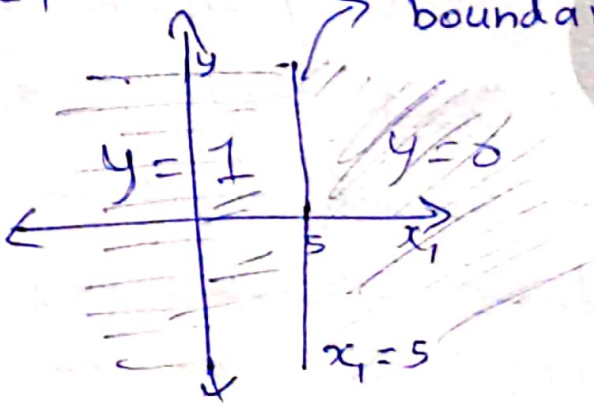
$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$

example :-

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$

$$\Rightarrow \theta^T x = 5 - x_1$$
$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\Rightarrow 5 - x_1 \geq 0 \Rightarrow y = 1$$
$$\Rightarrow x_1 \leq 5 \Rightarrow y = 1$$

plotting this, we get!-



decision boundary (*)

Decision boundary need not be a straight line

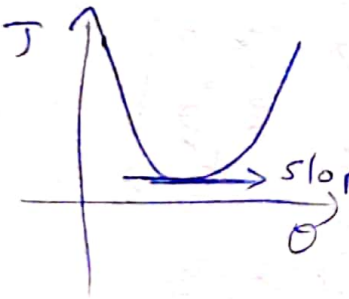
It was a straight line because we put $z = \theta_0 + \theta_1 x_1$. we can put anything in z, as long as we get a decision boundary that seperates $y=0$ & $y=1$ regions.

ex :- $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$ ⇒ circle shape

---

Alternative method to get minima ∴ Normal Equation.

Normal Equation :- Alternative to gradient descent.



$$\Rightarrow \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j} = 0$$

So, minima is directly found by calculating derivative of J, and equating it to 0.

Thus, we get required θ_j values.

Normal Equation Formula :-

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

, where

where $X = \begin{bmatrix} x_0 & x_1 & \cdots & x_n \\ x_0 & x_1 & & x_n \\ x_0 & x_1 & & x_n \\ x_0 & x_1 & & x_n \end{bmatrix}$ → ~~m rows~~ ~~m columns~~ m rows n+1 columns

and $x_0 = 1$

Gradient Descent	Normal Equation
* feature scaling sometimes necessary	no need of feature scaling
* ~~no~~ need to choose α	no need to chose α
* many iterations needed.	no iterations
* $O(kn^2)$ Time Complexity	$O(n^3)$, need to calculate $(X^T X)^{-1}$
Preferred method * for large n ($\approx \geq 10^4$)	Preferred if n is small ($\approx < 10^4$)

Note :- sometimes $X^T X$ may be noninvertible

Solution :- 1) delete redundant features. :-
i.e. closely related (Linearly dependant).

ex :- x_1 : feet
x_2 : metre

2) Too many features ($m \leq n$) :-
delete some features
(or)
use Regularization

Logistic Regression Model :

Cost Function :

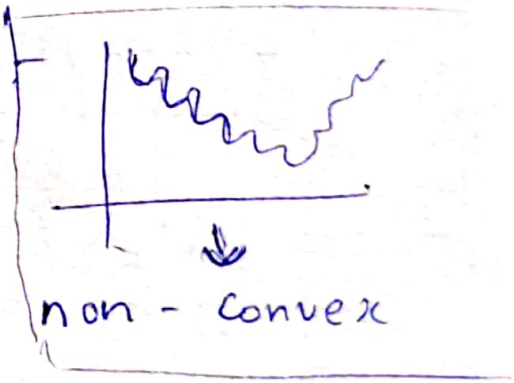
Linear Regression :- $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2}(h_\theta(x_i) - y_i)^2$

↳ let this be $\text{Cost}(h_\theta(x_i), y_i)$

so, for linear regression, cost function is :-

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

but, for logistic regression, this square-error formula cannot be used, because it has a lot of local minima, like this :-



plot of $\text{cost}(h_\theta(x), y)$, where $\rightarrow$

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

So gradient descent won't converge to global minimum.

In order to solve this problem, we have to make cost fn Concave :-

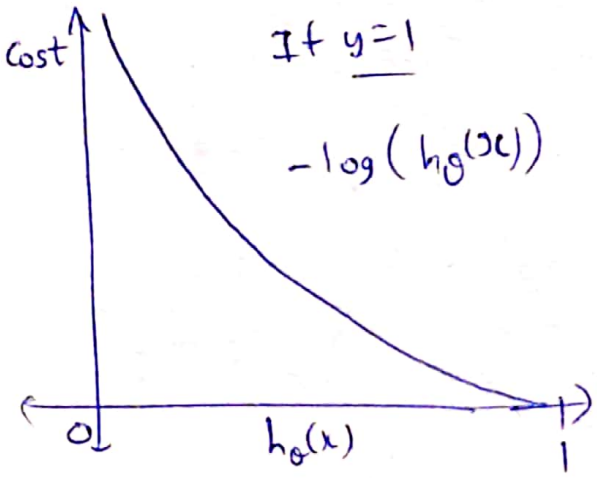
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

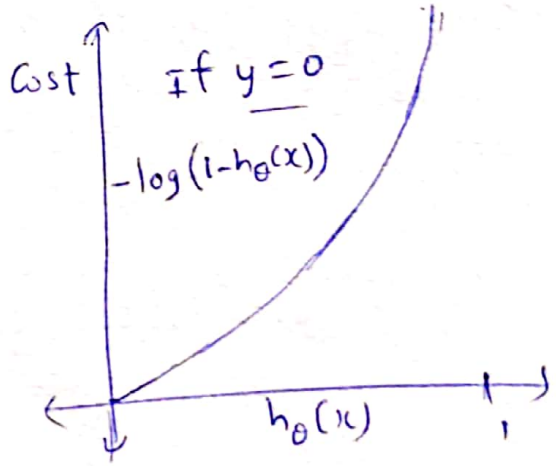
↓ one liner ↓

Logistic Regression's Cost Function :

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

This yields a concave function.





Advantages of this kind of cost function :-

~~Penalize the training set example hypothesis function if~~

Cost = 0, if $y=1$ & $h_\theta(x) = 1$
& if $y=0$ & $h_\theta(x) = 0$

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

This ensures that we penalize the learning algorithm by a large cost, if we get incorrect prediction.

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

↓

for logistic Regression -

Gradient Descent :-

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

}

upon taking Partial derivative, we get :-

Repeat :- {

$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i\right) x_{ij}$

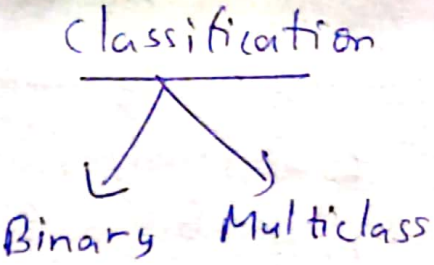
}

There are some faster ways of obtaining θ than gradient descent :-

1) Conjugate gradient
2) BFGS
2) L-BFGS.

Multiclass classification :-

classifying data into more than 2 categories.



i.e. $y \in \{0, 1, 2, \ldots n\}$

set $\rightarrow$ n+1 categories

---

how to classify ? :-

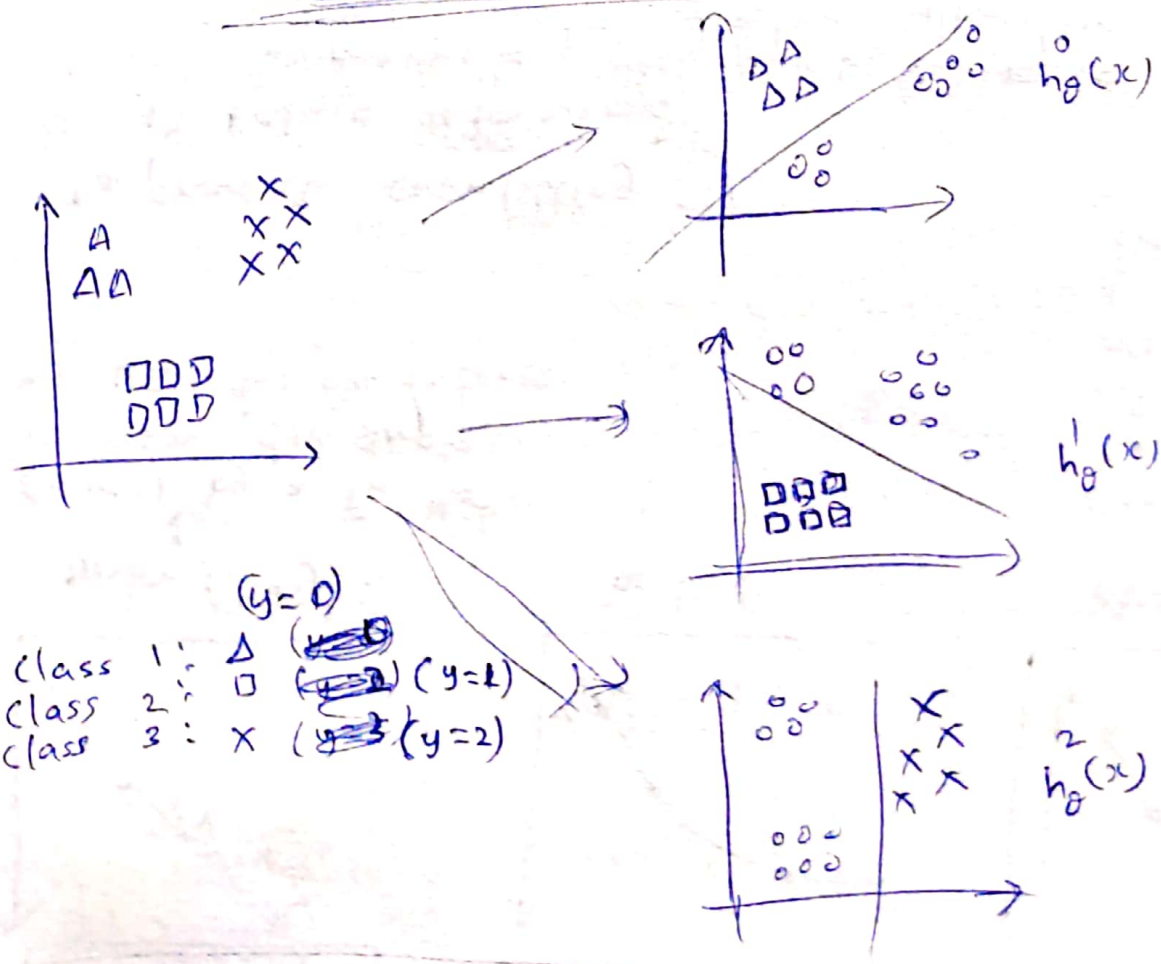
Steps :-

(1) Compute $h_\theta^i(x)$, $\forall\ i \in \{0, 1, 2, \ldots n\}$

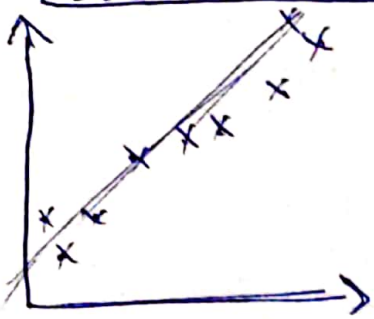
so, $h_\theta^i(x) = P(y = i \mid x; \theta)$ — (*)

(2) Prediction = $\max\left(h_\theta^0(x), h_\theta^1(x), \ldots h_\theta^n(x)\right)$

its corresponding class is the answer.



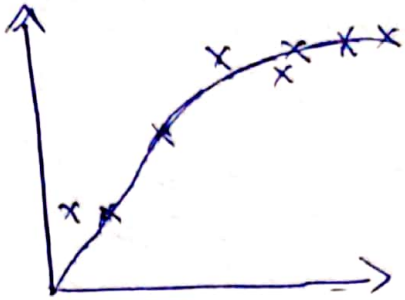
Overfitting :-



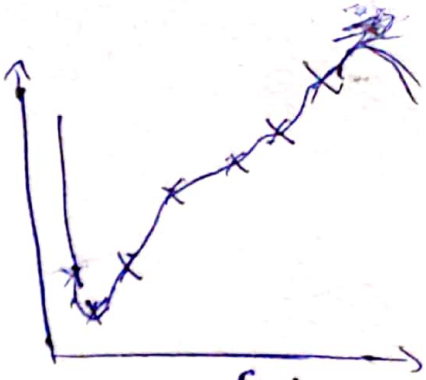
Under fitting

↓

Caused by a fn that is either too simple, (or) uses too few features.



best fit.



overfitting

↓

fits the given data very well but does not generalize well to predict new data.

To prevent overfitting :-

1) Reduce #features :-
   - → manually select which feature to remove
   - → use model selection algorithm.

2) Regularization :-
   - → keep all the features, but reduce the magnitude of parameters θ_j.
   - → Regularization works well when we have a lot of slightly useful features.

Regularization :-

This is done in order to prevent a few parameters dominating the remaining parameters

How do parameters dominate, in the first place? :- At the beginning of gradient descent, we will assume a hypothesis function & give some initial values to ~~each~~ each θ_j, by using

gaussian distribution. Then we run gradient descent, which will assign new values to each θ_j. Sometimes, few θ_j will have large values, compared to the remaining θ_j. This means domination.

In order to reduce domination, we penalize all ~~parameters~~ θ_j parameters, by adding a new term, called Regularization Term into the cost function. Regularization term also solves overfitting by ~~flattening~~ smoothening the curve, thereby making it more generalized.

When does overfitting occur :-

If we have a small data set ('m' is small), then we can easily come up with a curve to fit all values in the given data set. But it incorrectly predicts the new values.

Regularization steps :-

once you have fixed the parameters (area, length, weight, etc) and finalized some rough form of hypothesis fn, you can do regularization by using the following cost function :-

$$\min_\theta \frac{1}{2m} \sum_{j=1}^{m} \left(h_\theta(x_j) - y_i \right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

$\lambda \rightarrow$ regularization parameter.
↳ usually 0-1
↳ if it is too large, under-fitting may happen.

note that θ_0 is not ~~optimi~~ penalized.

Regularization Formulae :-

Linear Regression :-

Gradient Descent :-

Repeat :-
{

$$\theta_j := \theta_j\left(1 - \frac{\alpha\lambda}{m}\right) - \frac{\alpha}{m}\sum_{i=1}^{m}\left(\left(h_\theta(x_i) - y_i\right)x_{ij}\right)$$

}, $j \in \{0, 1, 2, \ldots n\}$

note :- (1) $1 - \frac{\alpha\lambda}{m} < 1$, always.

$\Rightarrow$ θ_j is reduced by small amount on every update.

(2) remaining term $\rightarrow$ no change.

Normal Equation :-

$$\theta = \left(X^T X + \lambda . L\right)^{-1} X^T y \quad \text{where } L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \end{bmatrix}$$

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

note :-

If $m < n$, $(X^T X)$ $\rightarrow$ non-invertible.

But if we add $\lambda . L$ term to it,

$(X^T X + \lambda L)$ $\rightarrow$ invertible (i.e. inverse is possible)

Logistic Regression :-

Cost function :-

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

Gradient Descent :-

Repeat {

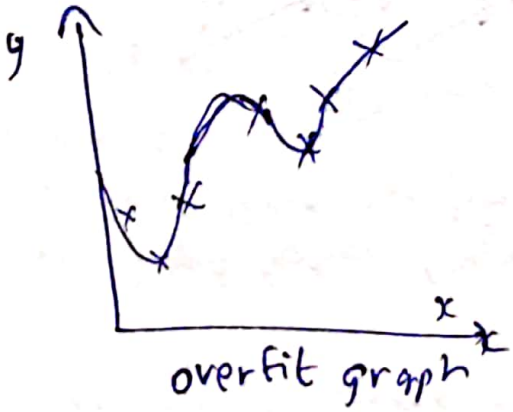
$$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{0i}$$

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{ij} + \frac{\lambda}{m}\theta_j\right]$$

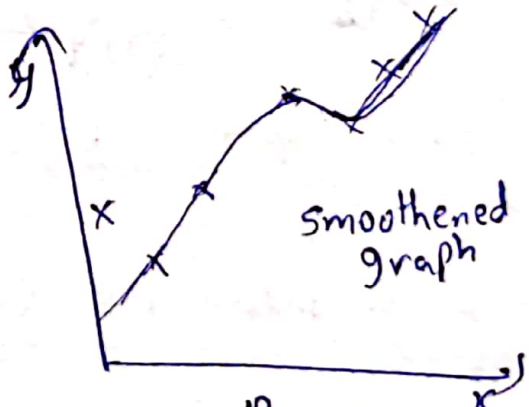
}

3

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



overfit graph



Regularized graph.

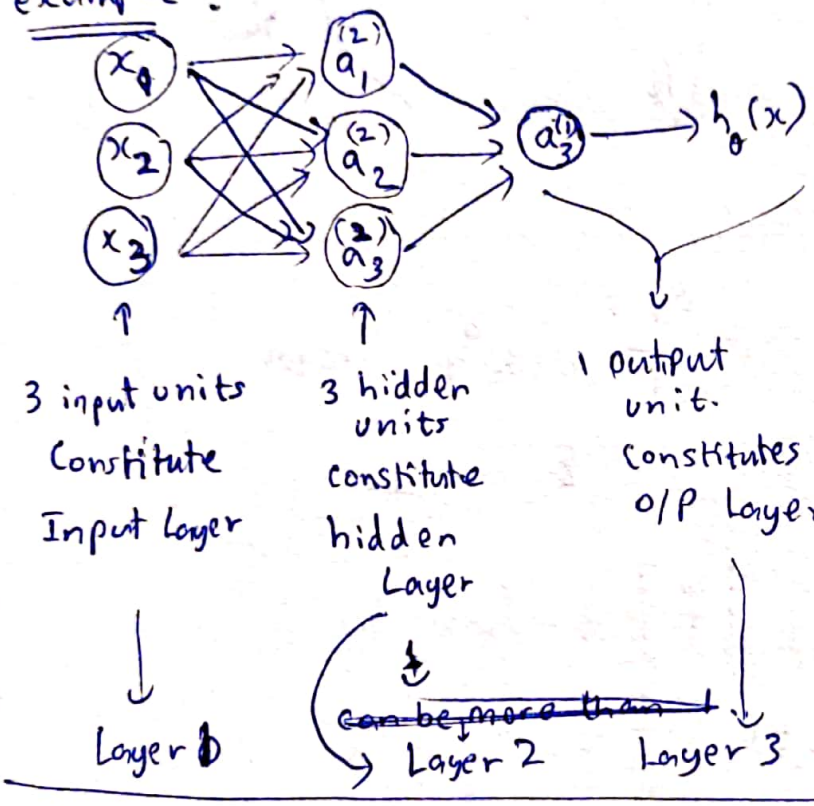
Neural Networks :-

Mimic the brain's learning ability.
electric

In each neuron, we have dentrites, which take inputs and axons, where electric pulses are sent (output).

In this model, dentrites are input features $(x_1, x_2, \dots x_n)$, and output is the result of our hypothesis function.

example :-



There can be more than 1 hidden / Intermediate Layer.

Hidden Layer
↓
a.k.a
↓
Intermediate Layer.

---

hidden layer units → denoted by $a_i^{(j)}$
(activation of unit i in layer j)

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

$\theta_{ab}^{(j)}$ → a^{th} row, b^{th} column of $\theta^{(j)}$.

θ^j → It has dimension: $(s_{j+1}) \times (s_j + 1)$, where ~~$s_j$ = # units in~~

where $s_j \rightarrow$ # units in layer j

$s_{j+1} \rightarrow$ # units in layer $j+1$

$x_0 \rightarrow$ "bias unit"

$\rightarrow$ always $= 1$

$\theta^{j}_{ab} \rightarrow$ "weights"

How to compute $h_\theta(x)$ for above example?

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

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

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

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

and, $h_\theta(x) = a_1^{(3)}$

$=$.

Here, $g \rightarrow$ sigmoid logistic activation function & $g(z) = \frac{1}{1+e^{-z}}$

$x_0 = 1$, $a_0^{(2)} = 1$