# Machine Learning

**Definition :**

① ML is a field of study that gives computers the ability to learn without being explicitly programmed [Arthur Samuel, 1959].

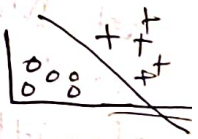
② A computer program is said to learn from experience E w.r.t some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

**ML Algorithms**

- Supervised learning
- Unsupervised learning

**Others:** Reinforcement learning, recommender systems.

① **Supervised learning** :- "right answers" given. labelled output.



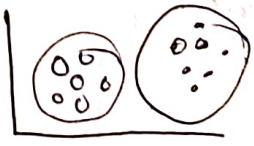
**Regression:** Predict continuous valued output.

**classification:** Discrete valued output (0 or 1)

② **Unsupervised learning**:- no label or same label of all data.

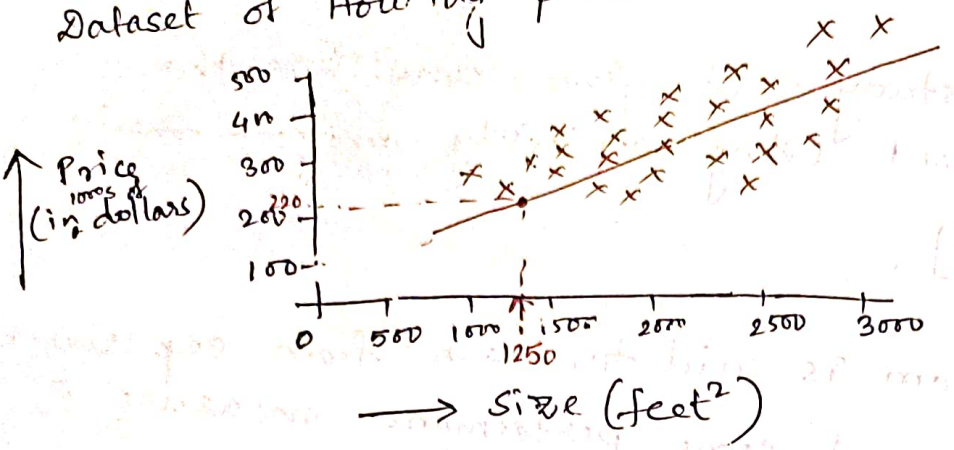
**clustering:**

- used to organize large computer clusters
- Applicⁿ on social network analysis.
- Market segmentation.



## ⑪ Linear Regression with one variable :⇒

Dataset of Housing prices.




A number of houses that were different sizes and sold for a range of different prices

For a house of size 1250 feet², how many prices will be ~~valued~~? considered?
– may be 220 dollars.

**Supervised**

Given the "right answer" for each example in the data

~~It~~ Most commonly used supervised algorithm is

**classification** that gives discrete valued output

**Regression**

Predict real-valued output.

Training set of housing prices.

| Size in feet² (x) | Price ($) in 1000's (y) |
|---|---|
| 2104 | 460 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ... | ... |

Let : m be the number of training examples.
x : i/p variables / features , y : "o/p variable / target variable".

Here, in our example, $m = 47$.

$(x, y)$ - one training example.

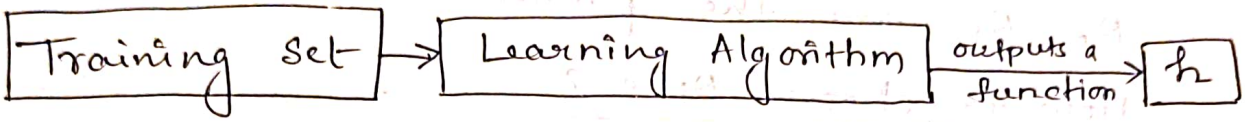
$(x^{(i)}, y^{(i)})$ - $i^{th}$ training example.

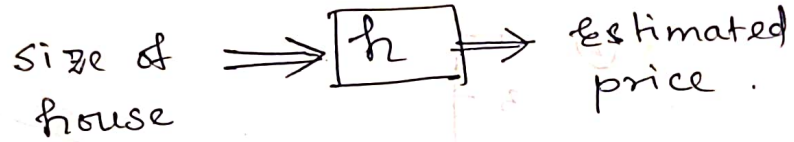
i.e., $x^{(1)} = 2104$

$x^{(2)} = 1416$

$y^{(1)} = 460$.

h stands for hypothesis.



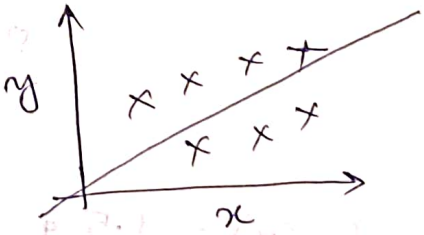

So, h is a function, (called hypothesis), that maps from x's to y's.

» How do we represent h?

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

$h_\theta(x)$ is also written as, $h(x)$.

It means that, we are going to predict that y is a linear function of x.




- Sometimes, linear function does not work. "non linear" has to be incorporated. But since linear case is the first (simple) building block, we start with this example of fitting a linear functions.

This model is called linear regression.

$h(x) = \theta_0 + \theta_1 x$ is linear regression with one variable. Predicting all the prices as functions of one variable $x$.

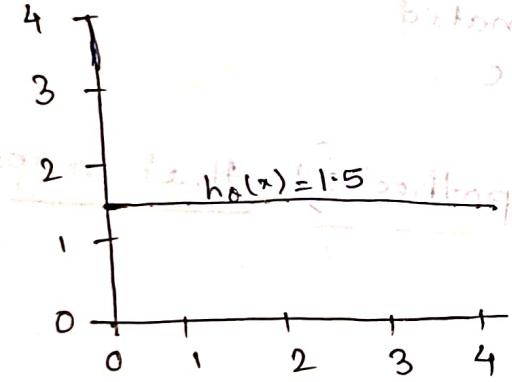
Also, called, univariate linear regression.

## ⊛ Cost function

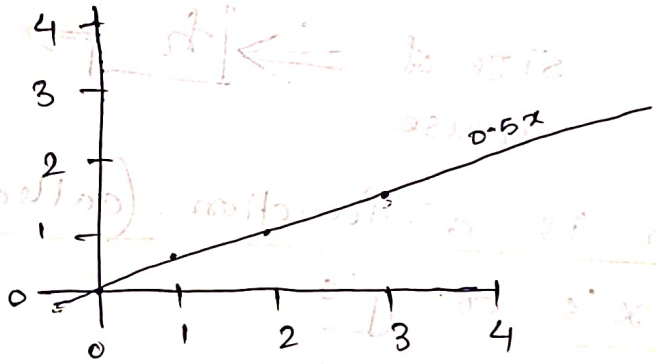
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

$\theta_0, \theta_1$ are the parameters of the model.

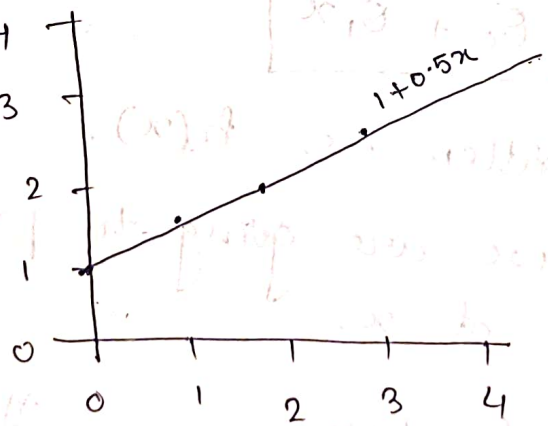
How to choose $\theta_i$'s ?

① 


$\theta_0 = 1.5$
$\theta_1 = 0$

② 

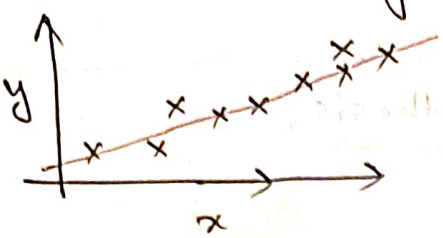

$\theta_0 = 0$
$\theta_1 = 0.5$

③ 


$\theta_0 = 1$
$\theta_1 = 0.5$

① $h_\theta(x) = 1.5$ ✱ , ② $h_\theta(x) = 0.5x$ ., ③ $h_\theta(x) = 1 + 0.5x$

» In linear regression, we have a training set.




we want to come up with values for the parameters $\theta_0$ & $\theta_1$ so that the straight line we get, corresponds to a straight line that somehow fits the data well (like in the above figure).

So, how to get such $\theta_0$ & $\theta_1$, that have good fit to data?

**Idea:** Choose $\theta_0$ & $\theta_1$ so that $h_\theta(x)$ is close to $y$ for our training examples $(x, y)$.

($y$ is the actual price), $h$ is the hypothesis).

$y$ & $h$ should be close to each other.

So, in linear regression, we want to solve a minimization problem.

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

where, $h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$,

⇓ This is the overall objective function for linear regression. (deciding $\theta_0, \theta_1$).

So, Cost function is,

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

$$\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1).$$

↓ cost function (also called, squared error function).

- The squared error func$^n$ is most commonly used cost function.

However, there are other alternative cost function as well.

Let us now use a simplified hypothesis,

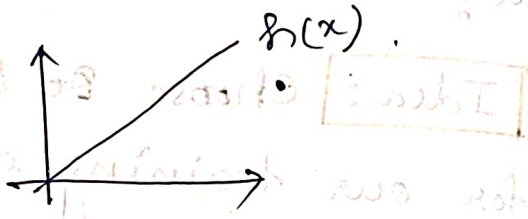
$h_\theta(x) = \theta_1 x$ , $\theta_0 = 0$

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

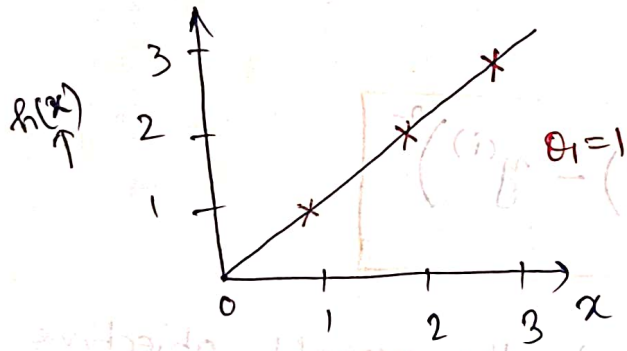
Goal:

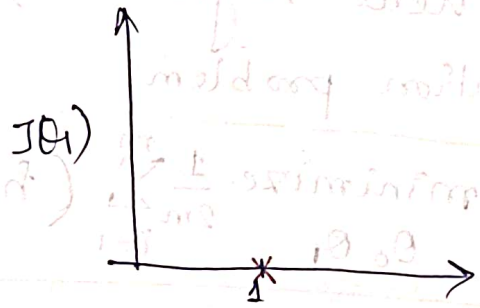
minimize $J(\theta_1)$
$\theta_1$

When, $\theta_0 = 0$, this means →




| $h_\theta(x)$ | $J(\theta_1)$ |
|---|---|
| (for fixed $\theta_1$, this is a func$^n$ of $x$) | (func$^n$ of the parameter $\theta_1$) |




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

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

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

Here, $x = y$ on the above graph.

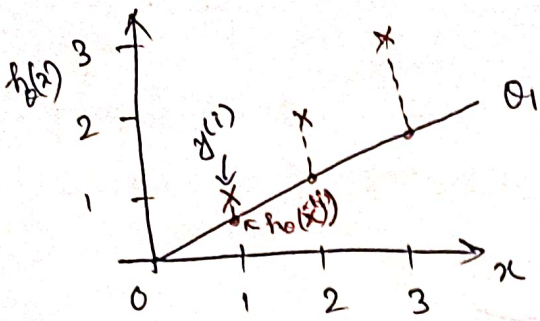
So, $J(\theta_1) = \frac{1}{2m} \times (0^2 + 0^2 + \cdots 0^2)$

i.e. $J(1) = 0$ | $= 0$, Here $m = 3$. 3 training examples: (1,1), (2,2), (3,3)

~~$\theta_1 = 0.5$~~

~~$J(0.5) = \frac{1}{2m}$~~

$\theta_1 = 0.5$.




∴ -

When $x = 1, y = 1$

& $h(x) = 0.5x$

$\therefore J(0.5)$

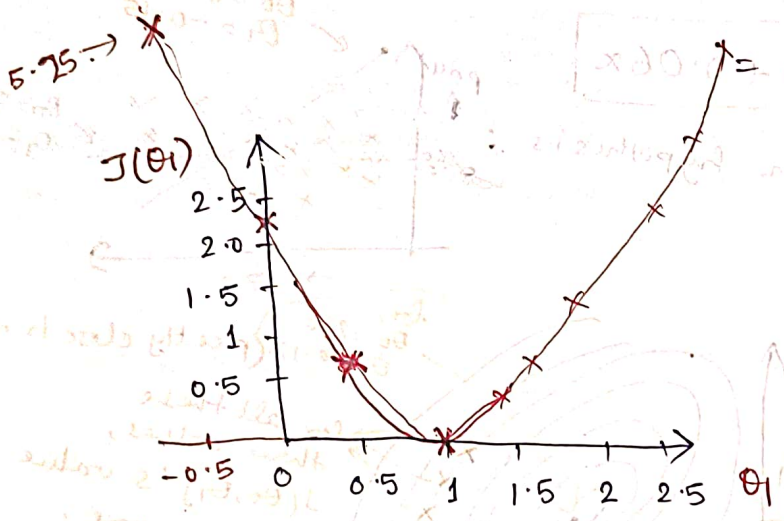
$$= \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$= \frac{1}{2m} \times \sum_{i=1}^{m} \left( 0.5x^{(i)} - y^{(i)} \right)^2$$

$$= \frac{1}{2m} \times \left[ (0.5-1)^2 + (1-2)^2 + (1.5-3)^2 \right]$$

$$= \frac{1}{2m} \times \frac{1}{2 \times 3} \times 3.5$$

$$= 0.58.$$




$\theta_1 = 1, J(\theta_1) = 0$

$\theta_1 = 0.5, J(\theta_1) = 0.58$

$\theta_1 = 0, J(\theta_1) = \frac{1}{6}[1^2 + 2^2 + 3^2]$

$= \frac{1}{6} \times 14 = \frac{14}{6} = 2.3$

$\theta_1 = -0.5x$

$J(\theta_1) = 5.25$

~~Our objective is~~ The optimization objective for our learning algorithm is, we want to choose the value of $\theta_1$ so that it minimises $J(\theta_1)$. ~~This is the~~

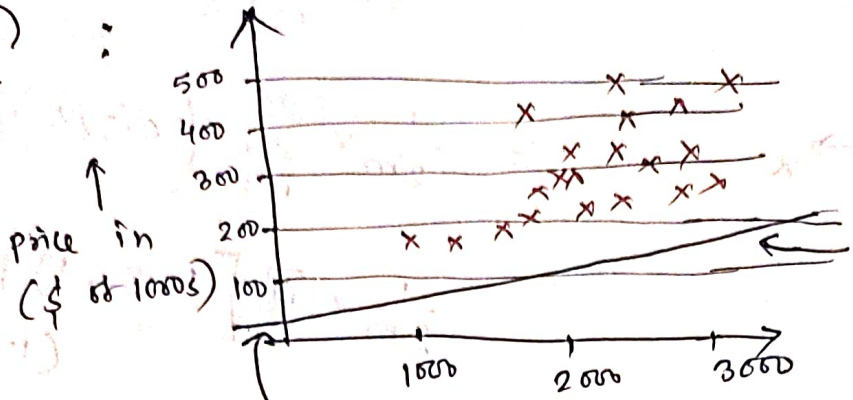
$$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$$

← This is our objective func$^n$ of linear regression.

Here, in the above example, the value that minimizes $J(\theta_1)$ is, $\theta_1 = 1$. That is the possible best line fit through our data.

>> Let us now keep both the parameters $\theta_0$ & $\theta_1$

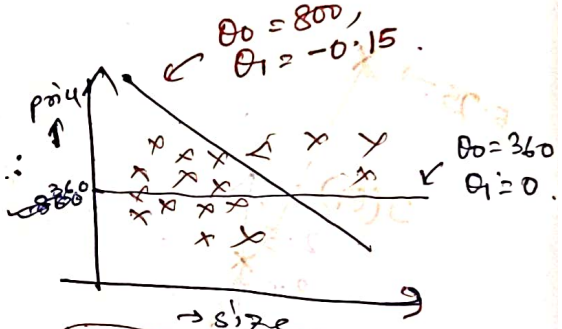
$h_\theta(x)$ :




housing price plot.

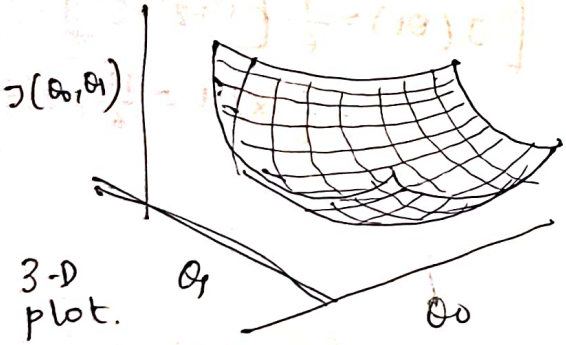
$\therefore \boxed{h_\theta(x) = 50 + 0.06x}$

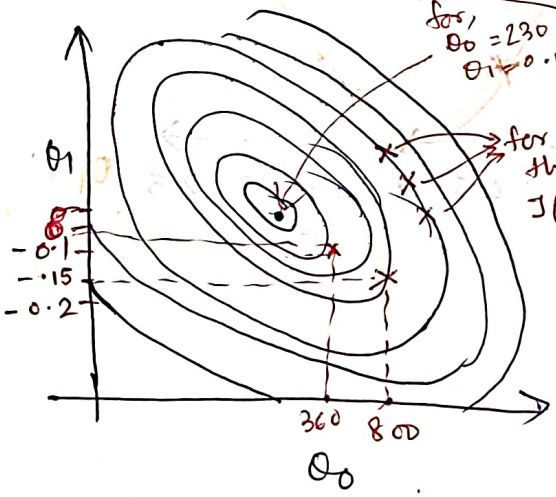
Another hypothesis :




Now,

$J(\theta_0, \theta_1)$:




Contour plot.

- Each of the ellipses ~~take sam~~ shows a set of points that takes on the same value ~~of θ0 & θ1~~ for $J(\theta_0, \theta_1)$.

⑥ Contour figure is a way (or a more convenient way) to ~~change~~ visualize my function J.

⑨ Example: $\theta_0 = 800$ & $\theta_1 = -0.15$ is not a very good fit to our example (as shown in the previous page). The value of $J(\theta_0, \theta_1)$ is pretty far from the minimum (i.e the centre) (as shown in the previous page).

i.e, pretty high cost of $J(\theta_0, \theta_1)$ we obtain.

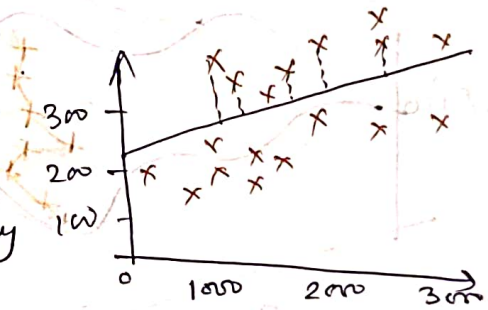
Again, if $\theta_0 = 360$, $\theta_1 = 0$. ($J(\theta_0, \theta_1)$ closer)

Again, ~~$\theta_0 = 500$, $\theta_1 =$~~ .

$\theta_0 = 230$, $\theta_1 = 0.12$, → $J(\theta_0, \theta_1)$ is pretty close to minimum.

[ smallest & centered elipse have same height,
next elipse has height greater than the central one,
" " " " " " the previous " & so on.]

[the centre of the smallest elipse has minimum value].




So, we need an algorithm or software to find automatically the value of $\theta_0$ & $\theta_1$, that minimizes cost function J.

⑩ The Algorithm: Gradient descent algorithm.

It is the most general algorithm to find out values of $\theta_0$, $\theta_1$ that minises $J(\theta_0, \theta_1)$. It is not only used in linear regression, but also is used all over the places of machine learning and helps in minimizing others function as well apart from the cost function.

⊙ We have some function $J(\theta_0, \theta_1)$.

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

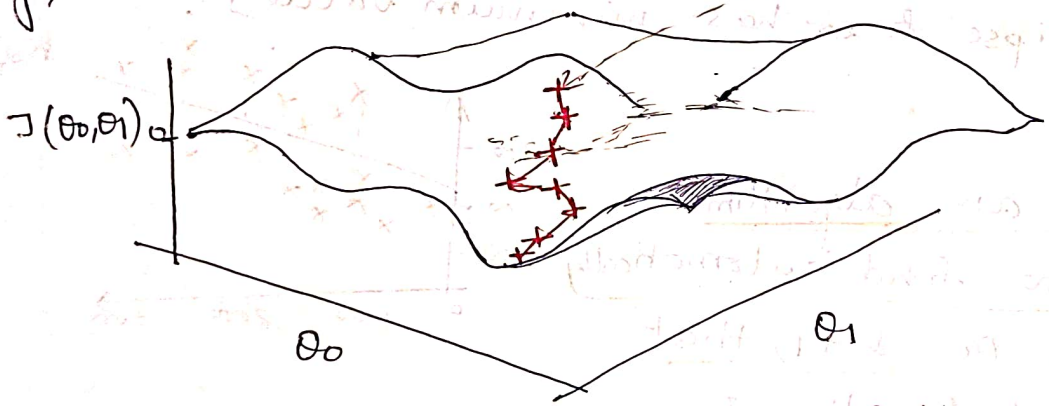
Outline:

⊙ Start with some $\theta_0, \theta_1$

⊙ Keep changing $\theta_0, \theta_1$ to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum.

[Gradient descent is actually applied to more general function, like $J(\theta_0, \theta_1, \theta_2, \dots \theta_n)$ and want $\min_{\theta_0, \theta_1, \dots, \theta_n} J(\theta_0, \theta_1, \dots, \theta_n)$ ]

Say, initialize $\theta_0 = 0$, $\theta_1 = 0$ ← starting point.




- imagine, you are standing on a hill.
- you want to ~~stop~~ go down.
- you look all around and take little baby step in some direction.
- you want to go downhill as quickly as possible.
- What direction do I take that little baby step in?
- you choose a direc$^n$ and go down and down until you get the local minimum.

- Similarly, by choosing another direction, you can reach to another local minimum (which is very different from the previous one).

Algorithm :

repeat until convergence {

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

}

- $\alpha$ is a number that is called learning rate.
- $\alpha$ denotes how big step we take downhill with creating descent.
- If $\alpha$ is very large, then it corresponds to a very aggressive gradient descent procedure where we are trying to take huge step downhill.
- Simultaneously update $\theta_0, \theta_1$.

i.e,

$$\text{temp } 0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp } 1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp } 0$$

$$\theta_1 := \text{temp } 1.$$

Incorrect:

$$\text{temp } 0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp } 0$$

$$\text{temp } 1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_1 := \text{temp } 1$$

⊙ Example: $\theta_0 = 1$, $\theta_1 = 2$.

$\theta_j := \theta_j + \sqrt{\theta_0 \theta_1}$ (for $j = 0$ & $1$)

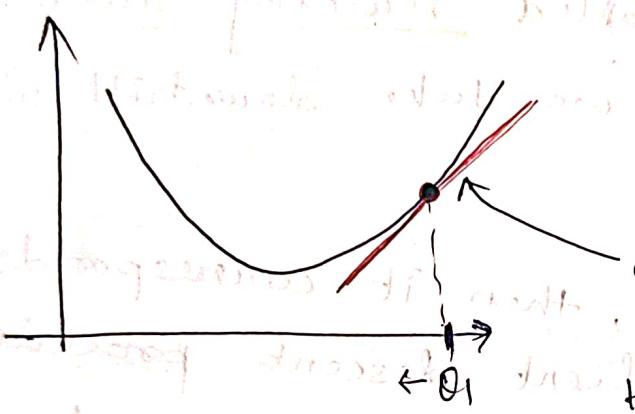
simultaneously update $\theta_0$ & $\theta_1$. What are the resulting values of $\theta_0$ & $\theta_1$?

Ans:

$\theta_0 = 1 + \sqrt{2}$

$\theta_1 = 2 + \sqrt{2}$

---

⊙ 


for simplicity (or ease of understanding), let's assume, we have only $\theta_1$.

derivative term denotes a slope of this line.

Here slope is positive.

So, $\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)}$ → this term is positive ($> 0$)

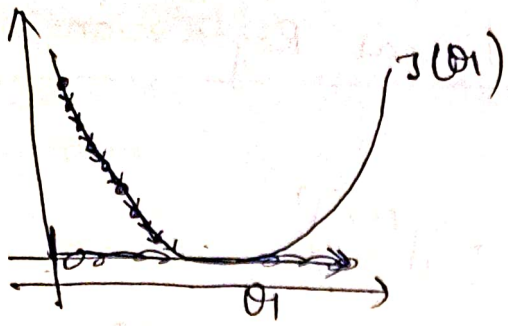
i.e, $\theta_1 := \theta_1 - \alpha$. (some positive value)

So, new updated $\theta_1$ will have less value compared to the previous $\theta_1$.

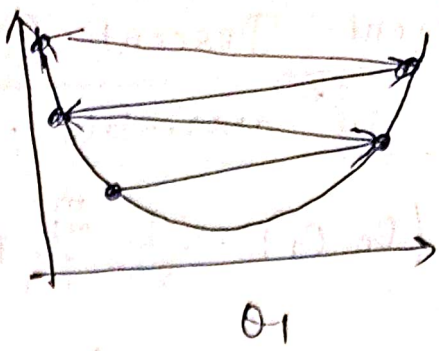
Similarly, if in some point, slope is negative then, updated $\theta_1$ will have increased value than the previous one.

⊙ If $\alpha$ is too small, gradient descent can be slow.

⊙ If $\alpha$ is too large, gradient descent can overshoot minimum, hence may fail to converge or diverge.




when $\alpha$ is too small

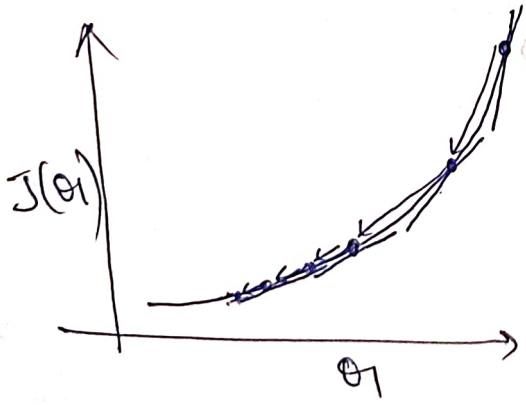



$\alpha$ is too large, GD takes huge step. It goes further away from minimum.

Qs:- what if the ~~local minimum is at~~ parameter $\theta_1$ is already in local minimum?

Ans: slope is horizontal in this case.
So, derivative value is 0. and $\theta_1 := \theta_1$ i.e value of $\theta_1$ remains unchanged.

② Even if the learning rate $\alpha$ is fixed, GD can converge to a local minimum. As we approach a local minimum, GD will automatically take smaller steps. So, no need to decrease $\alpha$ over time.




The less the stipness of the slope, the less the value of the derivative term. So, the smaller derivative term helps in taking the smaller steps (or baby steps) even if $\alpha$ is fixed.

## Gradient Descent (GD) for Linear Regression with one variable :-

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

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

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

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

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

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

And,

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

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

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

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

So, now,

Gradient descent algorithm for linear regression is,

repeat until convergence {

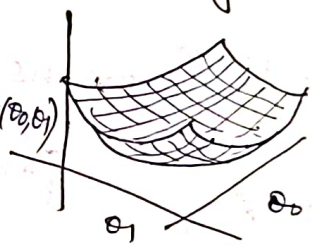
$$\theta_0 := \theta_0 - \alpha . \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha . \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right) . x^{(i)}$$

}

update $\theta_0$ & $\theta_1$ simultaneously.

① Cost function of linear regression is always going to be a bow shaped function.




The technical term of this function is, convex function.

Therefore, it always converges to the global optimum (there is no other local optimum).

② The GD algorithm, in ~~this case, is know~~ machine learning, is called Batch Gradient Descent.

"Batch": Each step of gradient descent uses all the training examples.

i.e, we use $\sum_{i=1}^{m}$ everytime when we update ~~calculate~~ $\theta_0$ or $\theta_1$.

⑤ House sizes:

2104
1416
1534
852.

$h_\theta(x) = -40 + 0.25x$

my matrix is, $\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}$.

and vector is $\begin{bmatrix} -40 \\ 0.25 \end{bmatrix}$.

$\therefore \begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 \\ 0.25 \end{bmatrix} = \begin{bmatrix} -40+0.25\times2104 \\ -40+0.25\times1416 \\ -40+0.25\times1534 \\ -40+0.25\times852 \end{bmatrix}_{4\times1}$

$h_\theta(2104)$.

$\therefore$ Prediction = Data matrix × Parameters

gives all the predictive prices.

$\gg \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix}_{2\times3} \times \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ .5 & 2 \end{bmatrix}_{3\times2}$

$= \begin{bmatrix} 1\times1+3\times0+2\times5 & 1\times3+3\times1+2\times2 \\ 4\times1+0\times0+1\times5 & 4\times3+0\times1+1\times2 \end{bmatrix}_{3\times2}$

$= \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}_{3\times2}$ Ans.

⑤ House sizes:

2104
1416.
1534
852.

Have 3 competing hypothesis:

1. $h_\theta(x) = -40 + 0.25x$
2. $h_\theta(x) = 200 + 0.1x$
3. $h_\theta(x) = -150 + 0.4x$

$\therefore \begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}_{4\times2} \times \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix}_{2\times3}$

$= \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}_{4\times3}$ = predicted housing prices.

$m = 4$, (4 houses). $\therefore$ 12 predictions we get.

$n = 3$, (3 hypothesis).

⑥ ~~Matrix Inverse~~:

If A is an $m \times m$ (square matrix) matrix, and if it has an inverse,

$$A.(A^{-1}) = A^{-1}A = I.$$

» Intuitively, if a matrix do not have any inverse, then it can be thought of as, it is too close to 0 in some sense.

» Matrices that don't have an inverse are "singular" or "degenerate" matrix.

## ① Linear Regression with multiple variables :-

⇒ for one variable :

| size (feet²) ($x$) | Price ($) ($y$) |
|---|---|
| 2104 | 460 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |

So, our hypothesis was, $h_\theta(x) = \theta_0 + \theta_1 x$.

» **Multiple features (variables)** :-

| size (feet²) ($x_1$) | # bedrooms ($x_2$) | # floors ($x_3$) | Age of home (years) ($x_4$) | Price ($1000) ($y$) |
|---|---|---|---|---|
| 2104 | 5 | 1 | 45 | 460 |
| 1416 | 3 | 2 | 40 | 232 |
| 1534 | 3 | 2 | 30 | 315 |
| 852 | 2 | 1 | 36 | 178 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

Notation:

$n$ : #features (here, $n = 4$)

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

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

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}, \quad \text{so, } x_3^{(2)} = 2.$$

**Hypothesis :**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \cdots + \theta_n x_n$$

For convenience, define $x_0 = 1$.

⑥ $$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\therefore h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T x$$

⇒ Multivariate Linear regression.

$$\underbrace{\begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix}}_{\theta^T} \times \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}}_{x}$$

» Gradient Descent for Linear Regression with multiple features :-

Hypothesis,

$$h_\theta(x) = \theta^T x = \theta_0 \overset{(x_0=1)}{x_0} + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Parameters, $\theta_0, \theta_1, \ldots, \theta_n$ (i.e $n+1$ dimensional vector $\theta$).

Cost function,

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

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

∴ Gradient Descent :

Repeat {

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

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

} (simultaneously update for every $j = 0, \ldots, n$)

## New Algorithm for $n \geq 1$ :

Repeat {

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

(simultaneously update $\theta_j$ for $j = 0, 1, \ldots n$)

}.

» Let us consider 2 features:

$x_0^{(i)} = 1$.

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

$$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_2^{(i)}$$

» It is important to choose features carefully.

Housing price prediction

$$h_\theta(x) = \theta_0 + \theta_1 \times \underbrace{\text{frontage}}_{x_1} + \theta_2 \times \underbrace{\text{depth}}_{x_2}$$

So, two features are there → $x_1$ & $x_2$.

It is not always necessary to use the features that are given. You can ~~use~~ create new features by yourself.

Let us create a new feature, as Area.

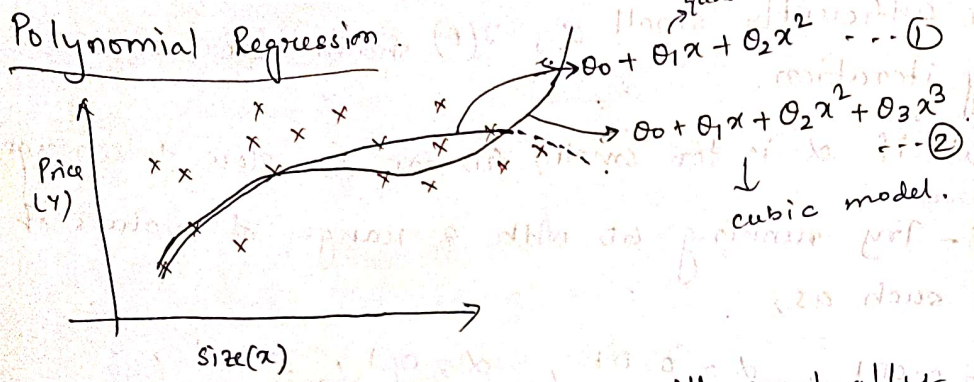
$x$ = frontage x depth = Area.

$\therefore h_\theta(x) = \theta_0 + x \cdot \theta_1$ ($x$ is the land area).

Sometimes, new features give better prediction.

Regression :

## ⑩ Polynomial Regression.




Note that, Eq^n ① gives the curve that will eventually decrease, thus housing price will decrease.

But, Eq^n ② gives the curve (just a cubic value $\theta_3 x^3$ is added to the Eq^n ①), that increases eventually

So, it is better to use Eq^n ②.

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

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

→ $x_1 = size$
$x_2 = (size)^2$
$x_3 = (size)^3$.

Also, we need scaling of $x_1$, $x_2$ and $x_3$.

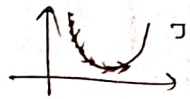
Also, $h_\theta(x)$ may be chosen as,

$h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 \sqrt{size}$. Then curve will not go down, but it will go up slowly.

So, we can use polynomial functions as well to fit our data.

» To solve the optimal value of parameter $\theta$, we sometimes use normal equation (instead of Gradient descent).

⇒ In G.D., we solve optimal value of $\theta$ by taking many steps, multiple iterations of G.D to converge to the global minimum.




⇒ In contrast, normal equation is a method to solve for $\theta$ analytically. So, instead of running iterative algorithm, we can just solve the optimal value for $\theta$ all in one go (i.e in one step).

Intuition: 0f. 1D ($\theta \in \mathbb{R}$)

cost func^n $J(\theta) = a\theta^2 + b\theta + c$ (assume, a quadratic func^n)

The way to minimize a quadratic function is by taking derivatives.

$$\frac{d}{d\theta} J(\theta) = \cdots \overset{set}{=} 0.$$

solve for $\theta$.

⇒ It is the simpler case when $\theta$ is just a real number.

⇒ But we are interested in the case $\theta \in \mathbb{R}^{n+1}$ where $\theta$ is not just a real number but it is a $n+1$-dimensional parameter vector, and

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

$$\therefore \frac{\partial}{\partial \theta_j} J(\theta) = \cdots \overset{set}{=} 0. \quad (\forall j)$$

solve for $\theta_0, \theta_1, \cdots, \theta_n$

Examples : $m = 4$

| $x_0$ | Size (feet²) ($x_1$) | # bedrooms ($x_2$) | # floors ($x_3$) | Age of home (years) ($x_4$) | Price ($1000) ($y$) |
|---|---|---|---|---|---|
| 1 | 2104 | 5 | 1 | 45 | 460 |
| 1 | 1416 | 3 | 2 | 40 | 232 |
| 1 | 1534 | 3 | 2 | 30 | 315 |
| 1 | 852 | 2 | 1 | 36 | 178 |

add an extra column.

$m = 4$ training examples.

matrix $X$ is all the features

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}_{m \times (n+1)}$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}_{m}$$

$m$ dimensional vector.

$\therefore \boxed{\theta = (X^T X)^{-1} X^T y}$ ⇒ This gives you the value of $\theta$ that minimizes the cost function.

(X is also called design matrix).

$$X^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2104 & 1416 & 1534 & 852 \\ 5 & 3 & 3 & 2 \\ 1 & 2 & 2 & 1 \\ 45 & 40 & 30 & 36 \end{bmatrix}$$ : OK

$$\therefore X^T X = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2104 & 1416 & 1534 & 852 \\ 5 & 3 & 3 & 2 \\ 1 & 2 & 2 & 1 \\ 45 & 40 & 30 & 36 \end{bmatrix} \times \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 2104+1416+1534+852 & 13 & 6 & 45+40+30+36 \\ 2104+1416+1534+852 & (2104\times2104)+(1416\times1416)+(1534\times1534)+(852\times852) & (2104\times5)+(1416\times3)+(1534\times3)+(852\times2) & (2104\times1)+(1416\times2)+(1534\times2)+(852\times1) & (2104\times45)+(1416\times40)+(1534\times30)+(852\times36) \\ 13 & \vdots & & & \end{bmatrix}$$

* Let,

| age ($x_1$) | height (cm) ($x_2$) | weight (kg) ($y$) |
|---|---|---|
| 4 | 89 | 16 |
| 9 | 124 | 28 |
| 5 | 103 | 20 |

predict a child's weight as a function of his age & height with the model.

$$\text{weight} = \theta_0 + \theta_1 \cdot \text{age} + \theta_2 \cdot \text{height}$$

$$\text{So, } X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, \quad y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$$

is this wrong? because, the ans. is given as, $X = \begin{bmatrix} 4 & 89 & 1 \\ 9 & 124 & 1 \\ 5 & 103 & 1 \end{bmatrix}_4$

$y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$. Why??

So,

$\theta = (X^T X)^{-1} X^T y$ is the optimal value of $\theta$ for which we get $\min_{\theta} J(\theta)$.

---

⑥ m training examples, n features.

| Gradient Descent | Normal Equation |
|---|---|
| i) Need to choose $\alpha$ | i) No need to choose $\alpha$. |
| ii) Needs many iterations. | ii) Don't need to iterate. |
| iii) Works well even when n is large. $[O(kn^2)$ time$]$. | iii) Need to compute $(X^T X)^{-1}$ $[O(n^3)$ time for $(n \times n)]$ $[\leftarrow O(n^3)$ time complexity for $n \times n]$. |
| $n = 10^6$, use G.D. | • Slow if n is very large. $[$inverse with $n = 1000$ is still ok, but, $n = 100000$, it is difficult$)$. In that case use G.D$]$. |

---

» Normal Equation for multivariate Linear Regression.

Normal Equ$^n$:

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

- What if $X^T X$ non-invertible? (Singular/degenerate)

(pinv is pseudo inverse).
inv = inverse

∴ ⇒ If $X^T X$ is non-invertible, there usually two most common causes for that.

---

① Cause i) If somehow, in your learning problem, you have redundant features.

Concrete, if you are tying to predict housing prices and if $x_1$ is the size of house (in feet$^2$) and $x_2$ is the size in meter$^2$, then $x_1$ and $x_2$ are linearly dependent $(x_1 = (3.28)^2 x_2)$.

⊛ In Linear algebra, one can see that if two features are related by some linear equation, then the matrix $(X^T X)$ is non-invertible.

Cause ii) If you are trying to run the learning algorithm with a lot of features.

Concretely, $m \leq n$.

Let $m = 10$, $n = 100$.

So, $\theta \in \mathbb{R}^{101}$. ⇒ i.e, you are trying to fit 101 parameters from just 10 training examples.

⇒ It is not a good idea.

In this case, either delete some features or use regularization.

# ⑪ Classification

## » Logistic Regression :⇒

Some examples of classification:

→ Email : spam / not spam ?

→ Online Transac" : Fraudulent / not Fraudulent?

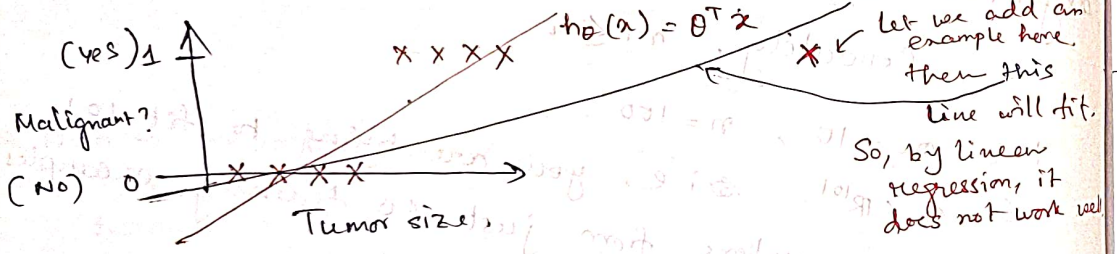
→ Tumor : Malignant / Benign ?

$y = \{0, 1\}$ 0: "Negative class" (e.g., benign tumor)

1: "positive class" (e.g., malignant)

Let us start for 2-classes only (0 & 1).
Later we'll discuss multi-class problem.

How do we develop a classifica" algorithm?




→ You can decide a threshold classifier output $h_\theta(x)$ as 0.5:

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

if $h_\theta(x) < 0.5$, predict "$y = 0$".

→ It may look like linear regression, in this example, even though it is a classification ~~pro~~ we are interested in.

So, Applying linear regression to a classification problem is not a good idea.

Classification: $y = 0$ or $1$

$h_\theta(x)$ can be $> 1$ or $< 0$ (if use Linear regression).

Logistic Regression has the following property,

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

⇒ Even though the term "regression" is present with "Logistic regression", still, Logistic Regression is a classification problem.

label $y$ is discrete value.

## ⑫ Hypothesis Representation of Logistic Regression.

We want $0 \leq h_\theta(x) \leq 1$.

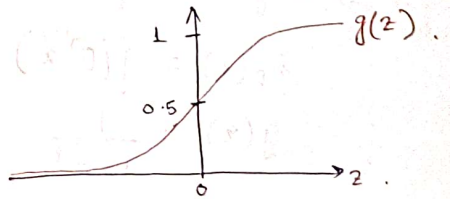
In Linear regression, $h_\theta(x) = \theta^T x$.

In Logistic " , $h_\theta(x) = g(\theta^T x)$, where

$$g(z) = \frac{1}{1+e^{-z}}$$ ⇒ this is called the sigmoid func" or Logistic ~~fu~~ func".

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

value of $g(z)$ is also between 0 and 1.



## » Interpretation of Hypothesis Output :

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

**Example:**

$$\text{If } x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumor size} \end{bmatrix}$$

$h_\theta(x) = 0.7$ .

Tell patient that 70% chance of tumor being malignant.

So, interpret as,

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

↓

"probability that $y=1$, given $x$, parameterized by $\theta$".

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

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

---

## ⓫ ~~Decision~~ Decision Boundary :-

How the ~~hy~~ hypothesis function looks like particularly when we have more than one features.

In Logistic Regression,

$$h_\theta(x) = g(\theta^T x) = P(y=1 \mid x ; \theta)$$

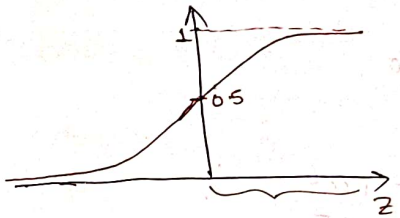
$$g(z) = \frac{1}{1+e^{-z}}$$

Suppose, predict "$y=1$" if $h_\theta(x) \geq 0.5$

$$\Rightarrow \theta^T x \geq 0$$

predict "$y=0$" if $h_\theta(x) < 0.5$ .

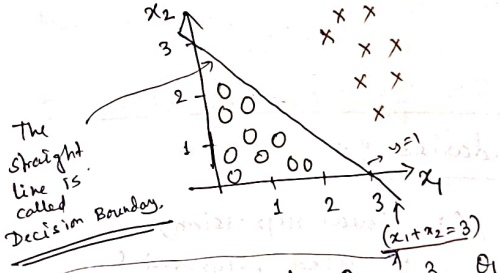
$$\Rightarrow \theta^T x < 0$$




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

as, $h_\theta(x) = g(\theta^T x) \geq 0.5$ when, $\theta^T x \geq 0$

### Decision Boundary :-




Let, we have a training set as shown in the figure and,

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

Let, values of $\theta_0 = -3$, $\theta_1 = 1$ and $\theta_2 = 1$ .

$$\therefore \theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

So, let us see when $y=0$ is being predicted & when $y=1$ is being predicted.

predict, "$y=1$" if $\underbrace{-3 + x_1 + x_2}_{(\theta^T x)} \geq 0$ . $\left[\because \theta^T x \geq 0 \text{ for } y=1\right]$.

$\Rightarrow x_1 + x_2 \geq 3$. ⇒ defines the equation of a straight line.

So, Decision Boundary is a property of the hypothesis, $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ (in this example).

(Not a property of the dataset)

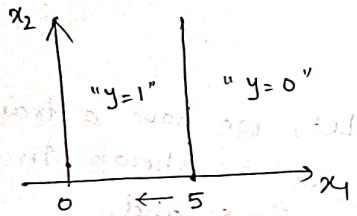
Example:

$\theta_0 = 5, \quad \theta_1 = -1, \quad \theta_2 = 0$.

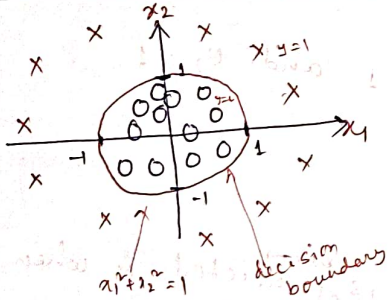
$h_\theta(x) = g(5 - x_1)$.

So, $h_\theta(x) \geq 0$ when $5 - x_1 \geq 0$

$\Rightarrow x_1 \leq 5$.




## » Non-linear decision boundaries :o ⇒




(in linear regression, we used polynomial regression where we added higher order terms).

Similarly, here also, we use 5 parameters.

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

Let, $\theta_0 = -1, \theta_1 = 0, \theta_2 = 0, \theta_3 = 1, \theta_4 = 1$.

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

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

$\Rightarrow x_1^2 + x_2^2 \geq 1$.

So, decision boundary looks like a circle.

## » Cost Functions of Logistic Regression:-

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

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

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

How to choose parameters $\theta$?

Cost func$^n$ of Linear Regression was,

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

$$= \frac{1}{m} \sum_{i=1}^{m} \text{cost}\left(h_\theta(x^{(i)}) - y^{(i)}\right).$$

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

$= \frac{1}{2}$ of squared error.

After simplifying, we get,

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

⇓

This is the cost I want my learning algorithm to pay if it outputs that value if its prediction is $h(x)$ and actual label was $y$.

For the Logistic Regression, if we would use the same cost function, we get non-convex function of the parameter's data.


for, $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ (non-linear func$^n$);

(many local optima).

(So, it is not guaranteed to converge to the global minimum).

('convex' is bow-shaped where it converges to global minimum, where is only one global minimum).

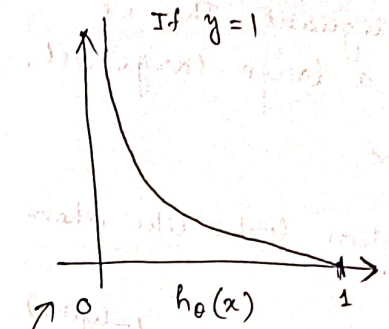
For the non-linear function $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$, $J(\theta)$ ends up being a non-convex function if you were to define it as a square cost function.

## Logistic Regression Cost Function :-

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

The cost or penalty that the algorithm pays, if it upwards the value of $h(x)$.

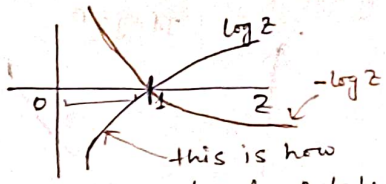
Let's plot the function:




$-\log(h_\theta(x))$

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

$0 \le h_\theta(x) \le 1$.




plot of $\log z$ is,

this is how log func$^n$ looks like...

we are interested only with the func$^n$ within range 0 to 1. Therefore, the plot looks like this:

Here, when $y = 1$,

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

cost $\to \infty$ if $h_\theta(x) \to 0$.

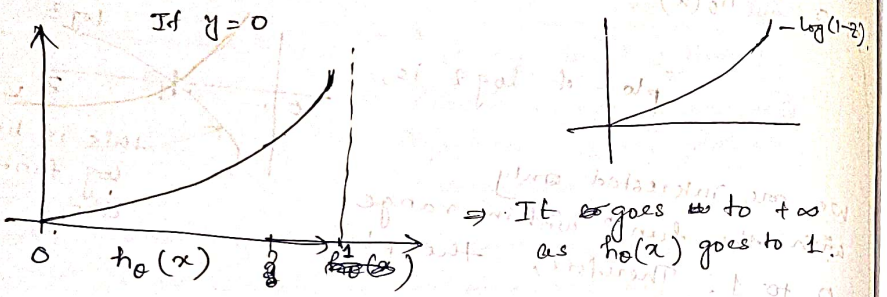
So, the hypothesis $h_\theta(x)$ with $y=1$ is basically $P(y=1|x;\theta)$.

Now, when we write $h_\theta(x) = 0$, this means, $P(y=1|x;\theta) = 0$ (when $y=1$). That is, we are saying that, say for a medical patient, probability of being a malignant patient is 0. This depicts a level of certainty, i.e., we are making it sure that the patient is not malignant. But, let the patient

, in actual, turns out to be malignant. Then we penalize the learning algorithm a large large cost.

So, $y=1$, $h_\theta(x) \to 0 \Rightarrow \text{cost} \to \infty$

Let's now see how the cost function looks like when $y=0$.




$\Rightarrow$ It goes to $+\infty$ as $h_\theta(x)$ goes to 1.

i.e, when we tell the hypothesis $h_\theta(x)=1$, i.e $P(y=0|x;\theta)=1$ with certainty, then also we end up with a large cost.

Similarly, when $y=0$ and

$h_\theta(x)=0$, cost $=0$.

---

## Cost Function of Logistic Regression

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

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

# For a classificaⁿ problem, y is always y = 0 or 1.

Concretely, we can write,

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

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

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

— This cost function is derived from statistics using the principle of maximum likelihood estimation, which is an idea in statistics of how to efficiently find parameters' data for different models. Also, it has a nice property that it is convex.

To fit parameters $\theta$:

$$\min_\theta J(\theta) \Rightarrow \text{Get } \theta.$$

To make a prediction given new $x$:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ with the values of $\theta$.

So, let's see how to minimize $J(\theta)$.

## Gradient Descent :⇒

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

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

Repeat {

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

(simultaneously update all $\theta_j$)

}

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

If you have $n$ features, then

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

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

» feature scaling makes gradient descent run faster.

---

## ⓫ Advanced optimization Idea

» Optimization Algorithm

Cost function $J(\theta)$. Want $\min_\theta J(\theta)$.

Given $\theta$, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \ldots, n$).

1. Gradient descent:

Repeat {

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

}

2. Conjugate gradient
3. BFGS
4. L-BFGS.

→ Advantages:
- No need to manually pick $\alpha$.
- Often faster than gradient desc

→ Disadvantage:
- More complex.

## Example: (how to use these algorithms)

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

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

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

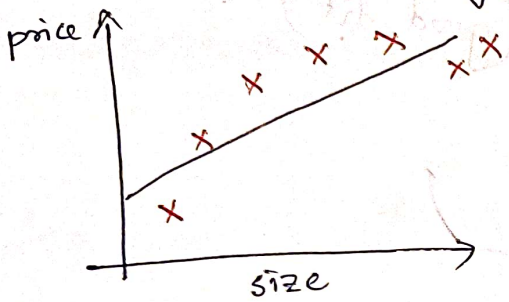
<use matlab code to use an optimization algorithm>.

$x_1 \leq 6$.

$x_2 = 0$ $x_1 = 6$

## ⑧ The problem of overfitting :-

Example: Linear Regression (housing prices).



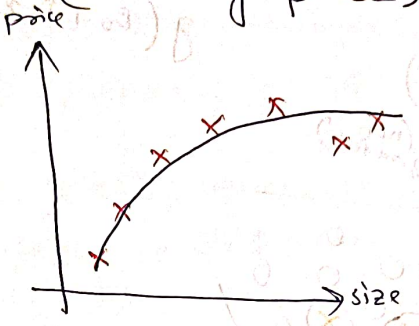

$\theta_0 + \theta_1 x$

(not a very good model)

"High bias"

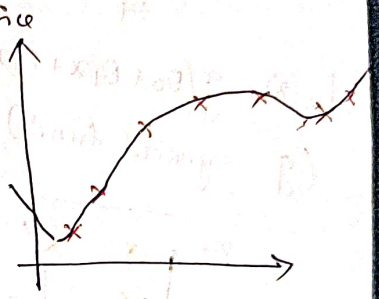
↓

does not fitting properly.




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

works well




$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

seems to have good job, but actually not.

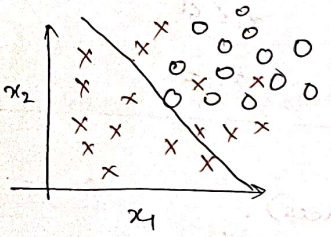
"overfitting".

"High variance".

The intuition is that, if we're fitting such a high order polynomial, then the hypothesis can fit as it almost fit any func$^n$ and this face of possible hypothesis is just too large, its too variable. and we do not have enough data to constrain it to give us a good hypothesis. so, that's called overfitting.

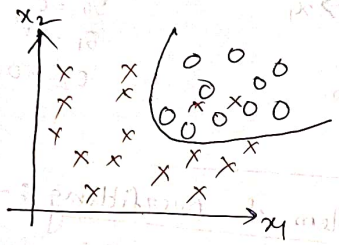
» The problem of overfitting comes when we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$) but fail to generalize to new examples (predict prices on new examples).

» Same thing can be applied for logistic regression as well.



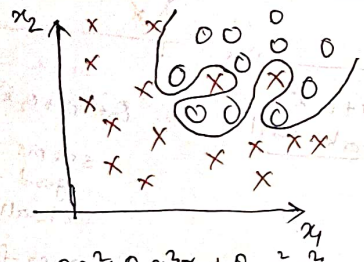

$h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2)$

($g$ = sigmoid func$^n$). (not good fit)




$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

(good fit)




$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \cdots)$

(over-fitting).

If overfitting is occuring, what we do to address it? we could just plot the hypothesis and see what was going on and select the appropriate degree polynomial.

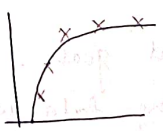
→ When we have so many ~~polynm~~ features, it becomes much harder to plot the data & also becomes harder to visualize. If you have lot of features but less training data, then overfitting is a common problem.

So,

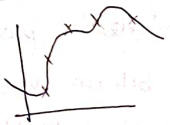
1. Reduce number of features
   - Manually select which features to keep.
   - Model selection algorithm.

2. Regularization
   - keep all the features, but reduce magnitude / values of parameters $\theta_j$
   - Works well when we have a lot of features, each of which contributes a bit to predicting $y$.

» Regularization :- Cost function :-



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



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$.

Suppose, we want to penalize and make $\theta_3, \theta_4$ really small.

$$\min_{\theta} \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

To minimize the new func$^n$, let, $\theta_3 \approx 0$ & $\theta_4 \approx 0$. So, this is now a quadratic func$^n$.

# Non-linear Hypothesis :

## Neural Networks

Consider a Non-linear classificaⁿ :-


Let, you have a training set like this. If you want to apply logistic regression, one thing you could do is,

apply logistic regression with lot of non-linear features like $g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$

But, this particular method works only when you have two features $x_1$ & $x_2$ and all the polynomial terms of $x_1$ & $x_2$ in $g$(which is the sigmoid func$^n$).

But, we may have lot more features than just two.

Let, in a housing problem,
$x_1$ = size
$x_2$ = #bedrooms
$x_3$ = #floors
$x_4$ = age
⋮
$x_{100}$

} 100 diff. features.

Let, we are taking an example of identifying a car from an image.

If we chose only two pixel areas of 50×50 pixel size of each region, then we have 50×50 = 2500 pixels,

i.e., $n = 2500$ (gray scale). (7500 if RGB).

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 "} \\ \vdots \\ \text{pixel 2500 "} \end{bmatrix}$$

Now, if we use Logistic regression in this case, ~~with~~ including all the quadratic terms ($x_i x_j$) as features, then there are total $x_i \times x_i \approx$ ~~3~~ 3 ~~features~~ million features.

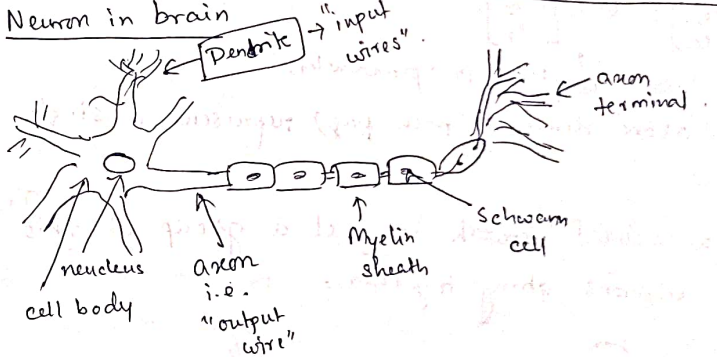
**Origins of Neural Network** :- → Algorithms that try to mimic the brain.

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

⇒

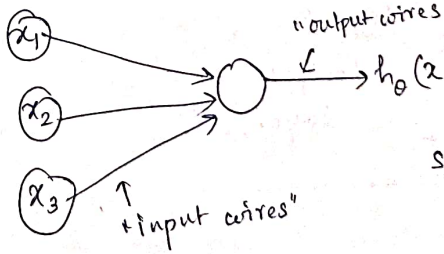
**Recent Resurgence** :- State-of-the-art technique for many applications as computers become faster. NN has complex algorithm.

Neuron in brain




Neurons communicate with each other by little pulse of electricity.

» Neuron Model : Logistic unit




Sometimes, $x_0$ is also drawn. $x_0$ is called the "bias unit". because, $x_0$ is already equal to 1.

» "Activation func" is a terminology used in Neural Network which is a term or function of non-linearity

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

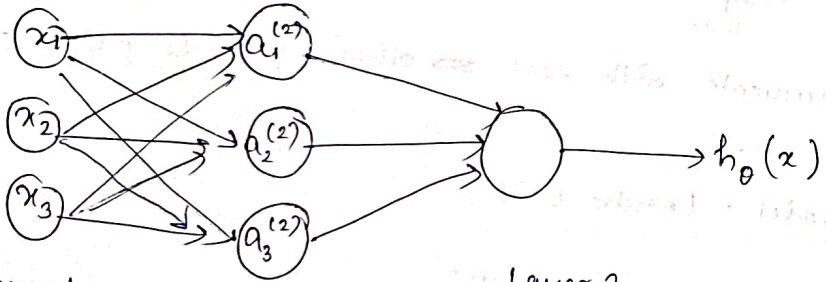
» 'Weight' of a model is same thing as the parameters (i.e., $\theta_0, \theta_1, \theta_2, \ldots$) of a model.

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

↑ "weights" or "parameters"

So, the above diagram (prev. page) represent a single neuron.

» what a neural network is, just a group of this different neurons strong together.




Layer 1 neurons $x_1, x_2, x_3$ (sometimes we add $x_0$ also)

Layer 2 neurons ($a_1, a_2, a_3$) (also we may add $a_0$).

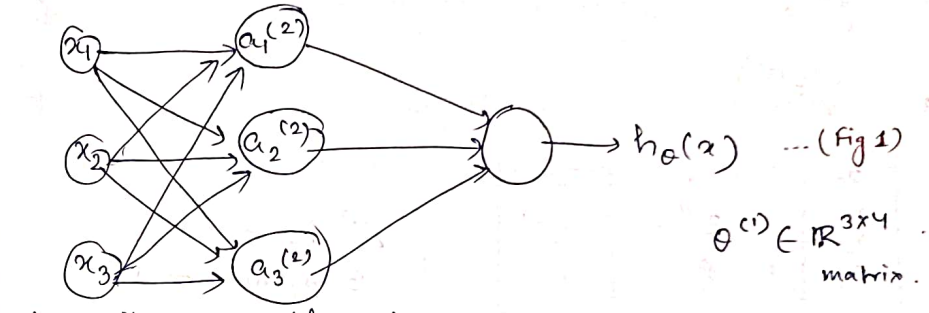
Layer 3. (output layer)

Layer 1 is input layer

Layer 2 is hidden layer.

» a hidden layer of values you don't get to observe in the training setup.

In the above example, we have only one hidden layer. But there can have more than one hidden layer.




...(Fig 1)

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

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

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

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

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

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

[sigmoid func" or sigmoid activation function" also called logistic activation function].

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

(one unit in 3rd layer).

(*) » If network has $s_j$ units in layer $j$, $s_{j+1}$ units in layer $j+1$, then $\theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

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

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

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

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

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

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$, $z^{(2)}$ is a 3-dimensional vector.

$x$: feature vector. $z^{(2)}$: vector of z-values.

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

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

$a^{(2)} \in \mathbb{R}^3$ (i.e., 3-dimensional vector) $z^{(2)} \in \mathbb{R}^3$ (i.e., a 3-dimensional vector).

Here, $g$ is the activation function of the elements of $z^{(2)}$s.

We can consider,

$a^{(1)} = x$.

So, $a^{(1)}$ is activation's input layer.

There is a 'biased unit' in every layer.

So, add $a_0^{(2)} = 1$ in $a^{(2)}$.

$z^{(3)} = \theta^{(2)} a^{(2)}$. $\boxed{h_\theta(x) = a^{(3)} = g(z^{(3)})}$

» We start of the propagation from input units and then we sort of forward-propagate that to the hidden layer and compute the activations of the ~~hidden layer and forward propagate that and comp.~~ hidd output layer.

So, Input layer → hidden layer → Output layer.

So, we call it as Forward Propagation.

---

The hidden layer ~~con~~ of neural network computes logistic regression and get the hypothesis function

~~$h_\theta(x) = g(\theta_1^{(2)})$~~

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

The only difference from logistic regression is that, in logistic regression, we use $x_1, x_2, x_3$ directly instead of $a_0^{(2)}, a_1^{(2)}, a_3^{(2)}$. That means, layer 2 is learning by their own from layer 1 (by taking features of layer 1), and computes $h_\theta(x)$.

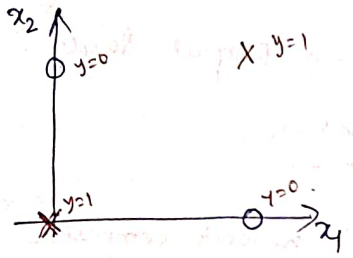
---

Network architecture means how the ~~net~~ neurons are connected to each other.

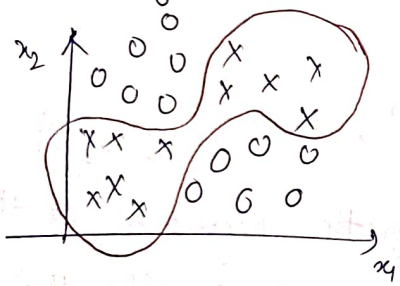
## » EXAMPLE 1 :-

Non-linear classification example : XOR / XNOR.

$x_1, x_2$ are binary (0 or 1).




Let we have ~~positive~~ bunch of positive examples (denoted by 'X') and negative examples (denoted by 'O').




So, our target is to learn a non-linear decision boundary that seperates the positive and the negative examples.

$y = x_1$ XOR $x_2$

and ~~or~~.

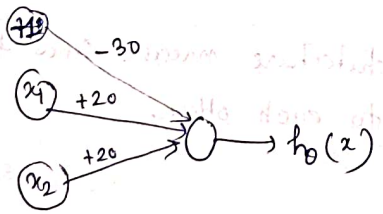
or, $y =$ ~~XOR~~ $x_1$ XNOR $x_2$

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

Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND $x_2$



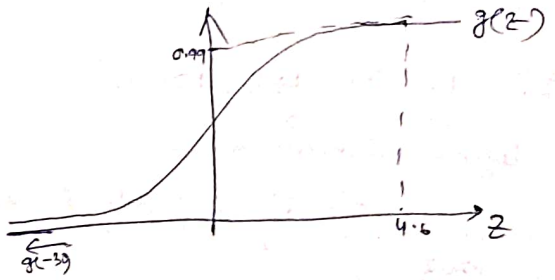

Can we get a one unit network to compute this logical AND operation? In order to do so, I am going to draw the bias unit. (i.e. +1 unit).

Let the parameters are, $-30$, $+20$, $+20$ resp.

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

where $\theta_{10}^{(1)}$, $\theta_{11}^{(1)}$, $\theta_{12}^{(1)}$

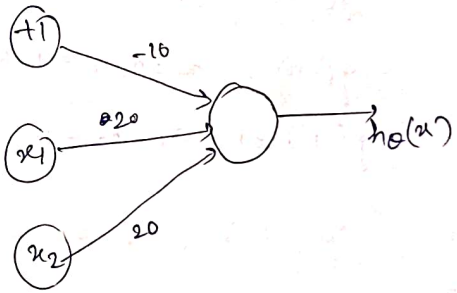
| $x_1$ | $x_2$ | $h_\theta(x)$ |
|---|---|---|
| 0 | 0 | $g(-30) \approx 0$ |
| 0 | 1 | $g(-10) \approx 0$ |
| 1 | 0 | $g(-10) \approx 0$ |
| 1 | 1 | $g(10) \approx 1$ |




$\therefore h_\theta(x) \approx x_1$ AND $x_2$

Logical AND that our neural network computes.

ex2 :-




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

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

$= g(-10) \approx 0$, when $x_1, x_2 = 0$

$= g(10)$, when $x_1 = 0$ & $x_1 = 1$

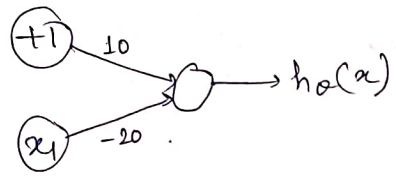
$= g(10)$, " $x_1 = 1$ $x_2 = 0$.

$= g(30)$ " $x_1 = x_2 = 1$

So, it is $x_1$ OR $x_2$.

(Ans)

Ex 3: Negation




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

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

Ex 4: (NOT $x_1$) AND (NOT $x_2$).

logical function is,

= 1 if and only if $x_1 = x_2 = 0$

AND

| $x_1 x_2$ | $(\sim x_1)$ | $(\sim x_2)$ | |
|---|---|---|---|
| 1 1 | 0 | 0 | 0 |
| 1 0 | 0 | 1 | 0 |
| 0 1 | 1 | 0 | 0 |
| 0 0 | 1 | 1 | 1 |

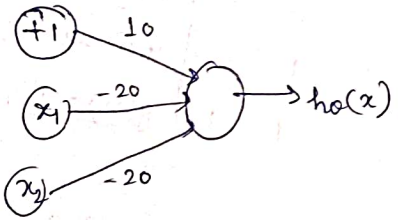
$h_\theta(x) = g(10 + -20x_1 + -20x_2)$

$= g(-30) \approx 0$ . , $x_1 = x_2 = 1$

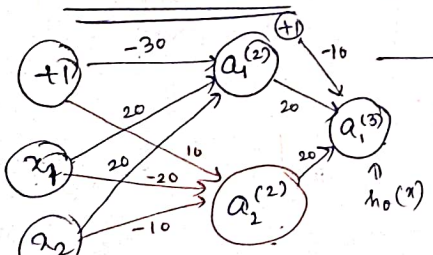
$= g(-10)$ , $x_1 = 1, x_2 = 0 \approx 0$.

$= g(-10)$ , $x_1 = 0, x_2 = 1 \approx 0$.

$= g(10) \approx 1$, $x_1 = x_2 = 0$ .




∴ $x_1$ XNOR $x_2$ : Putting $x_1$ AND $x_2$, (NOT $x_1$) AND (NOT $x_2$) and $x_1$ OR $x_2$ together.




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

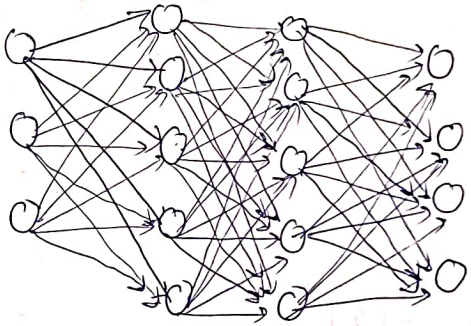
∴ $x_1$ XNOR $x_2$

= ($x_1$ AND $x_2$) OR [(NOT $x_1$) AND (NOT $x_2$)]

$a_1^{(2)}$ , $a_2^{(2)}$ ⇒ hidden layer

$a_1^{(3)}$ ⇒ output layer.

## Multiclass classification in Neural Network



Let we have 4 output units to classify :- pedestrian, Car, Motorcycle & Truck.

$h_\theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ pedestrian, $h_\theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ → car

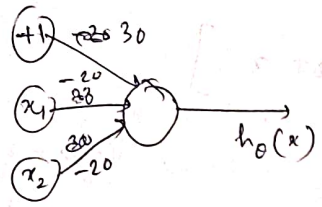
$h_\theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ motorcycle etc.

Hand written digits : multiclass classification

Training Set : $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})$.

$y^{(i)}$ is one of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ pedestrian, $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ car, $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ Motorcycle & $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ Truck. $\in \mathbb{R}^4$ (4 dimensional vector).

Assignment :-




| $x_1$ | $x_2$ | $h_\theta(x)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

OR

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

$x_1 = 0, x_2 = 0 \Rightarrow g(-20) \approx 0$

$x_1 = 0, x_2 = 1 \Rightarrow g(10) \approx 1$

$x_1 = 1, x_2 = 0 \Rightarrow g(10) \approx 1$

$x_1 = x_2 = 1$

$g(40) \approx 1$

30 - 20 = 10

30 - 40

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

$-20 + 30x_1 + 30x_2$

$\theta^{(1)} = \begin{bmatrix} 1 & 0.5 & 1.9 \\ 1 & 1.2 & 2.7 \end{bmatrix}$ $(a^{(2)})$

$\theta^{(2)} = \begin{bmatrix} 1 & -0.2 & -1.7 \end{bmatrix}$ $a^{(3)}$

| | |
|---|---|
| 0 0 | 0 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 1 |

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

$= g(\theta^{(1)} a^{(1)})$

$-(1.7+.2) = -1.9$

$= g(\theta_{10}^{(1)} a_0^{(1)} + \theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)})$

$= g\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} a_0^{(1)} + \begin{bmatrix} 0.5 \\ 1.2 \end{bmatrix} a_1^{(1)} + \begin{bmatrix} 1.9 \\ 2.7 \end{bmatrix} a_2^{(1)}\right)$

$= g(a_0^{(1)} + a_0^{(1)} + 0.5 a_1^{(1)} + 1.2 a_1^{(1)} + 1.9 a_2^{(1)} + 2.7 a_2^{(1)})$.

$h_\theta(x) = g(1.6 a_0^{(2)} - .2 a_1^{(2)} - 1.7 a_2^{(2)})$

$= g(a_0^{(1)} - 1.7 a_1^{(1)} - .2 a_2^{(2)})$ (after swapping).

## » Cost function of Neural Network :-

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

L = total no. of layers in network

$s_l$ = no. of units (not counting bias unit) in layer $l$.

### Binary classification

$y = 0$ or $1$

1 output unit

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

$s_L = 1$

⇓

#units in final layer $= K$

### 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

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

#output units = K.

### Cost function :-

#### Logistic Regression :-

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

#### Neural Network :-

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

$\Theta_{ji}^{(l)} \Rightarrow \Theta_{i0}^{(2)} x_0 + \Theta_{i1}^{(2)} x_1 + \cdots$

[বায়াস unit, বায়াস feature + other bias unit].

## Back Propagation Algorithm :⇒

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

The '$\delta$' term is, in some sense, going to capture the error in activation func$^n$ of that node, or so how we might wish the activation of that node was slightly different.

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

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

[$y_j$ is the actual value observed in our training example].

So, $a_j^{(4)}$ can also be written as,

$$(h_\Theta(x))_j$$

So, $\delta$ is the difference b/w hypothesis & actual value.

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

Here, each of $\delta^{(4)}$, $a^{(4)}$ & $y$ ~~are each of~~ is a vector whose dimension is equal to the number of output units in our network.

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

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

→ element wise multiplication operation.

Here, $g'(z^{(3)})$ formally is actually the derivative of the activation function $g$ evaluated at the input values given by $z^{(3)}$.

After calculating using calculus, we get,

$$g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)}).$$

$a^{(3)}$ is the vector of activations
1 is the n of 1s.

Similarly,

$$g'(z^{(2)}) = a^{(2)} .* (1 - a^{(2)}).$$

⇒ (*) There is no $\delta^{(1)}$ term, as the 1st layer is the input layer and the features we observed in our training set that does not have any error associated with it.

⇒ (*) So, we start computing the '$\delta$'-term from the output layer, and go back to the previous layers. So, we call it as back propagation.

⇒ (*) Though mathematically complicated, but the following is possible to prove.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$ [ignoring regularization term $\lambda$; or if $\lambda = 0$].

[Next continued after Decision tree; after 3-4 pages].

## » Bias, Variance, Overfitting & Underfitting

⇒ **Bias:** refers to distance between predicted value & actual value, i.e., how far we have predicted the values from actual values.

⇒ **High Bias:-** If the average prediction values are far away from actual values.

⇒ **Low Bias:-** Distance between predicted & actual values are minimal.

Let, $y = f(x) + e$, e is error.

Expected squared error at point x is,

$$Err(x) = E\left[(y - \hat{f}(x))^2\right]$$

$$= E\left[(y - f(x)) + (f(x) - \hat{f}(x))^2\right]$$

$\hat{f}(x)$ is predicted value.

y is original value.

$$= E\{(y - f(x))^2 + (f(x) - \hat{f}(x))^2 + 2[y - f(x)][f(x) - \hat{f}(x)]\}$$

$$= E[(y - f(x))^2] + E[(f(x) - \hat{f}(x))^2] + 2[E[y] - E[f(x)]$$

$$E[y \cdot f(x)] - E[y \cdot \hat{f}(x)] -$$

$$E[f^2(x)] + E[f(x) \cdot \hat{f}(x)].$$

$f(x)$: true func$^n$
$\epsilon$ : Error.
~~$\hat{y}$ = expected value from dataset D.~~
$\hat{y}$ = predicted value of target variable.
$h_D$ = predicted value from many dataset D.

$$f \equiv f(x) + \epsilon$$

$$\hat{y} = \hat{y}_D \equiv h_D(x)$$

∴ Expected squared error is,

$$E_{D,\epsilon}\{(f(x) + \epsilon - h_D(x))^2\} \quad \text{--- (1)}$$

$E_{D,\epsilon}$: dataset, noise
$f(x)$: true func$^n$
$\epsilon$: noise
$h_D(x)$: learned from D.

$$h \equiv E_D\{h_D(x)\} \Rightarrow$$ long term expectation of learner's prediction on this $x$ averaged over many datasets D.

(1) can be written as,

$$E_{D,\epsilon}\{(f - \hat{y})^2\}$$

$$= E\{([f - h] + [h - \hat{y}])^2\}$$

$$= E\{[f-h]^2 + [h-y]^2 + 2[f-h][h-y]\}$$

$$= E\{[f-h]^2 + [h-y]^2 + 2[fh - f\hat{y} - h^2 + h\hat{y}]\}$$

$$= E[(f-h)^2] + E[(h-\hat{y})^2] + 2(E[fh] - E[f\hat{y}] - E[h^2] + E[h\hat{y}])$$

$$= \underbrace{E[(f-h)^2]}_{\text{Squared difference between best possible prediction for } x, \text{ i.e., } f(x) \text{ and our long-term expectation for what the learner will do if we averaged over many datasets } D, \text{ i.e., } E_D[h_D(x)].} + \underbrace{E[(h-\hat{y})^2]}_{\text{Squared difference between our long-term expectation for the learners performance, } E_D[h_D(x)], \text{ and what we expect in a representative run on a dataset } D \text{ (i.e., } \hat{y})}$$

$\Downarrow$

$\text{Bias}^2$.

$\Downarrow$

Variance.

So, Error can be further ~~processed as~~ decomposed to,

$$\text{Err}(x) = \underbrace{\text{Bias}^2 + \text{Variance}}_{\text{reducible error.}} + \text{Irreducible Error.}$$

⊙ Variance is the variability of the model that how much it is sensitive to another subset of the training dataset.

If $y$ is actual value.

$\hat{y}$ is predicted value of target variable.

Then,

$$\text{variance} = E[(\hat{y} - E[\hat{y}])^2]$$

(or, $(h - \bar{y})^2$)

$\Downarrow$

the expected value of the square of the difference between predicted values and the expected value of the predicted values.

---

Continued Neural Network ...

Backpropagation Algorithm :⇒

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

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

this is used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

$\Theta$ is

These $\Delta$ deltas (capital) are going to be used as accumulators that will slowly add things to in order to compute these partial derivatives.

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

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

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

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

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{, if } j = 0 \text{ (when we do not have regularization term).}$$

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$$ [the formal proof is pretty complicated].

This is back propagation algorithm.

---

## Clustering

In clustering, we would like to have an algorithm that automatically group the data into coherent subset (i.e., coherent clusters for us).

» K-means clustering is, by far, the most popular clustering algorithm.

» Partitional Clustering :⇒

It is a type of clustering that devides the dataset into non overlapping subsets (or partitions) such that each data point is included in exactly one subset.

example : K-means clustering, k-medoids, Fuzzy c-means etc.

@ K-means Clustering :- An algorithm to achieve partitional clustering.

It aims to partition the data into K-clusters in which each data point belongs to the cluster with the nearest mean (center of the cluster), serving as a prototype of the cluster.

1. Initialize centroid in each cluster
2. Calculate Euclidian distance of a point to the centroid, for each cluster
3. Find min of the calculated value.
4. Update the centroid by taking mean
5. Follow step 2 to 4.

Example:

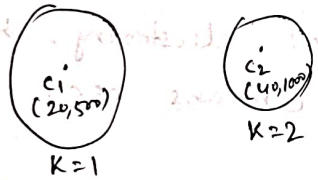
| Sl. no. | age | amount |
|---|---|---|
| $c_1$ | 20 | 500 |
| $c_2$ | 40 | 1000 |
| $c_3$ | 30 | 800 |
| $c_4$ | 18 | 300 |
| $c_5$ | 28 | 1200 |
| $c_6$ | 35 | 1400 |
| $c_7$ | 45 | 1800 |

Let $K=2$.

i.e., two clusters we'll create.

Step 1:

initialize centroids to two clusters.




Step 2:

for $c_3$ in K1,
(30,800)

distance $d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$ [If there is more parameters, then we again add the squared difference in the same way].

$= \sqrt{(30-20)^2+(800-500)^2}$

$= 300.\ldots$

for $c_3$ is k2,

$d = \sqrt{(30-40)^2+(800-1000)^2} = 200.24$.

As, $200.24 < 300\ldots$, So,

$c_3$ is in $K=2$ cluster.

Step 3: update centroid of $K=2$ cluster.

Centroid of $K=2$ cluster is,

$\left(\frac{40+30}{2}\right), \left(\frac{1000+800}{2}\right)$

$= (35, 900)$


Step 4:

for $c_4$ in K1, calculate d.

for $c_4$ in K=2, " d.

find less.

put in the corresponding cluster for which we get the lesser value.

update centroid.

continue the process.

---

>> Hierarchical Clustering: ⇒

Instead of dividing the dataset into a pre-determined number of clusters at once, hierarchical clustering builds up a hierarchy of clusters either through a divisive method (top-down approach) or an agglomerative method (bottom-up approach).

← It helps in making relation between two clusters.

## Agglomerative Clustering:-

- Initially consider every data point as an individual cluster and at every step, merge the nearest pairs of the cluster. (bottom - up).

~~- At first, every dataset.~~

Steps:

1. Calculate the similarity of one cluster with all the other clusters (calculate proximity matrix or other distance metric).
2. Consider every data point as an individual cluster.
3. Merge the clusters which are highly similar or close to each other.
4. Recalculate (update) the proximity matrix for each cluster.
5. Repeat steps 3 & 4 until only a single cluster remains.

Example:

Let, we have the data points,

1, 2, 3, 4, 5.

Lets create a distance matrix between every pair of units that you want to cluster.

A distance matrix will be symmetric (because, the distance between x and y is same as the distance between y & x). and will have zeroes on the diagonal (because, every item is distance zero from itself).

distance matrix ⇒

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | | | | |
| 2 | 9 | 0 | | | |
| 3 | 3 | 7 | 0 | | |
| 4 | 6 | 5 | 9 | 0 | |
| 5 | 11 | 10 | 2 | 8 | 0 |

minimum is 2 i.e, merging of 3 & 5. new cluster is '35'. Remove '3' & '5' & replace it by '35'.

Since we are using complete linkage clustering (a method of agglomerative clustering), the distance between "35" and every other item is the maximum of the distance between this (item & 3) and (item & 5). for example,

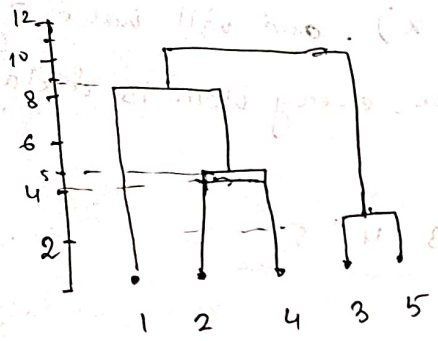
~~max d(1,3)~~ distance between "35" & "1" is ⇒

$$\max(d(1,3), d(1,5)) = \max(3, 11) = 11.$$

So, now, we have to update the distance matrix.

| | 35 | 1 | 2 | 4 |
|---|---|---|---|---|
| 35 | 0 | | | |
| 1 | 11 | 0 | | |
| 2 | 10 | 9 | 0 | |
| 4 | 9 | 6 | 5 | 0 |

Continuing this way, after 6 steps, everything is clustered.




This is called dendogram (a tree-like structure of clustering)

On this plot, the y-axis shows the distance between the objects at the time they were clustered. This is called cluster height.

Eg: cluster height of 35 is 2.

Let us now see, single linkage dendogram (below) for the same distance matrix.

It also starts from '35' but the distance between "35" and each item is now the minimum of $d(2,3)$ & $d(2,5)$. ie, $c(1, "35") = 3$.


Single Linkage.

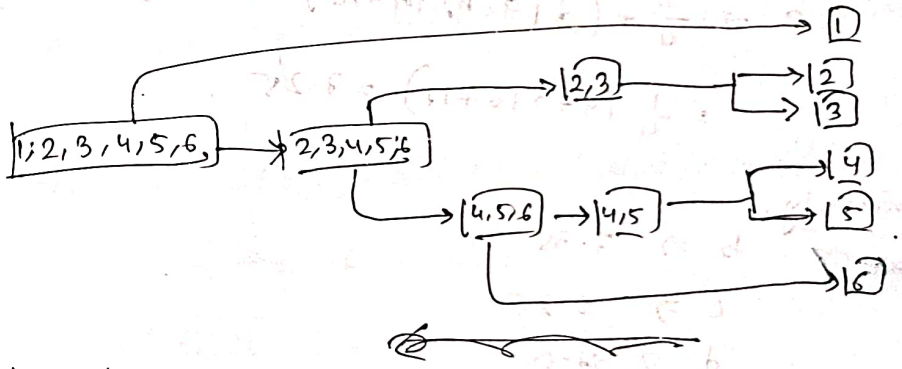
## Divisive clustering:-

- Consider all the data points as a single cluster.

Step1: Split into clusters using any flat-clustering method, say K-means.

Step2: Choose the best cluster among the clusters to split further, choose the one that has the largest Sum of Squared error.

Step3: Repeat steps 2 & 3 until a single cluster is formed.




top-down approach

## Example of Divisible clustering

Given dataset with distance matrix:-

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 9 | 3 | 6 | 11 |
| b | 9 | 0 | 7 | 5 | 10 |
| c | 3 | 7 | 0 | 9 | 2 |
| d | 6 | 5 | 9 | 0 | 8 |
| e | 11 | 10 | 2 | 8 | 0 |

Step 1: Initially $C_1 = \{a, b, c, d, e\}$.

Step 2: $C_i = C_1$ and $C_j = \phi$.

Step 3: Initial iteration

Let us calculate the average dissimilarities of the objects in $C_i$ with the other objects of $C_i$.

- Average dissimilarity of a is,

$$a \Rightarrow \frac{1}{4} * (d(a,b) + d(a,c) + d(a,d) + d(a,e))$$

$$= \frac{1}{4}(9+3+6+11) = 7.25.$$

Similarly,

$b \Rightarrow 7.75$

$c \Rightarrow 5.25$

$d \Rightarrow 7.00$

$e \Rightarrow 7.75$

- Take item of highest dissimilarity. Here b & e. choose anyone, say, b.
- We move b ~~from $C_j$~~ to $C_j$.
- we now have,

$C_i = \{a, c, d, e\}$ & $C_j = \{b\}$.

Step 4: Remaining iterations.

$C_i = \{a, c, d, e\}$, $C_j = \{b\}$

(i) 2nd iteration :-

$$D_a = \frac{1}{3}(d(a,c) + d(a,d) + d(a,e)) - \frac{1}{1}(d(a,b))$$

$$= \frac{20}{3} - 9 = -2.33.$$

$$D_c = \frac{1}{3}(d(c,a) + d(c,d) + d(c,e)) - \frac{1}{1}(d(c,b))$$

$$= \frac{14}{3} - 7 = -2.33$$

$$D_d = \frac{1}{3}(d(d,a) + d(d,c) + d(d,e)) - \frac{1}{1}(d(c,b))$$

$$= 0.67$$

$$D_e = \frac{1}{3}(d(e,a) + d(e,a) + d(e,d)) - \frac{1}{1}(d(e,b))$$

$$= \frac{21}{3} - 7 = 0.$$

Here, $D_d$ is the largest and >0.

So, move d to $C_j$.

we now have,

$C_i = \{a, c, e\}$,

$C_j = \{b, d\}$.

(ii) 3rd iteration :-

$$D_a = \frac{1}{2}(d(a,c) + d(a,e)) - \frac{1}{2}(d(a,b) + d(a,d))$$

$$= -0.5$$

$D_c = \frac{1}{2}(d(c,a)+d(c,e)) - \frac{1}{2}(d(c,b)+d(c,d))$

$= -13.5$

$D_e = \frac{1}{2}(d(e,a)+d(e,c)) - \frac{1}{2}(d(e,b)+d(e,d))$

$= \frac{13}{2} - \frac{18}{2} = -2.5$

All are negative, so we stop and from the clusters $C_i$ & $C_j$.

$C_i = \{a, c, e\}$ and $C_j = \{b, d\}$.

**Step 5:** To divide $C_i$ & $C_j$, we compute their diameters.

$\text{diameter}(C_i) = \max\{d(a,c), d(a,e), d(c,e)\}$

$\text{diameter}(C_i) = \max(3, 11, 2) = 11$

$\text{dia}(C_j) = \max\{d(b,d)\} = 5$.

As, $11 > 5$, consider $C_i$ cluster as the initial cluster & start dividing $C_i$ and repeat the same process.

**Bias:-**

Bias is the difference between the average predict of our model and the expected value which we are trying to predict.

Variance tradeoff
overfitting
underfitting
K-NN
SVM
RF

**Variance:-** Variance is the variability in the model prediction, meaning how much the predict's will change if a different training dataset is used.

## K-Nearest Neighbor

1. Calculate Distance
   (You can use Euclidian distance)
   $\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$
2. Select K-nearest neighbor. (You can tune K-value for getting good result).
   (K is the number of nearest neighbors).
3. Majority voting.

## Distances

1. Euclidian: $d(P,Q) = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$
2. Manhattan: $d(P,Q) = |x_2-x_1| + |y_2-y_1|$
3. Minkowski: $d(P,Q) = \left(\sum_{i=1}^{n} |x_i-y_i|^p\right)^{1/p}$

If $P=2$, it becomes Euclidian.
If $P=1$, " " Manhatten.

Example:

| Brightness | Saturation | class |
|---|---|---|
| 40 | 20 | Red |
| 50 | 50 | Blue |
| 60 | 90 | Blue |
| 10 | 25 | Red |
| 70 | 70 | Blue |
| 60 | 10 | Red |
| 25 | 80 | Blue |

Qsn is,

| Brightness | Saturation | class |
|---|---|---|
| 20 | 35 | ? |

1. Calculate distance ( say, Eucledian).

$d1 : \sqrt{(40-20)^2 + (20-35)^2} = 25.$

$d2 : \sqrt{(50-20)^2 + (50-35)^2} = 33.54.$

$d3 : \sqrt{(60-20)^2 + (90-35)^2} = 68.01.$

$d4 = 10$

$d5 = 61.03$

$d6 = 47.17$

$d7 = 45.$

2. Arrange them (the distances) in ascending order:

| Brightness | Saturation | class | Distance |
|---|---|---|---|
| 10 | 25 | Red | 10 |
| 40 | 20 | Red | 25 |
| 50 | 50 | Blue | 33.54 |
| 25 | 80 | Blue | 45 |
| 60 | 10 | Red | 47.17 |
| 70 | 70 | Blue | 61.03 |
| 60 | 90 | Blue | 68.01 |

Let's choose $K = 5$. So, choose the first five rows of the above table.

For 5-nearest neighbor, the majority class is 'Red'.

So,

| 20 | 35 | Red |
|---|---|---|

» How to choose K?

1. choosing a very low value will most likely lead to inaccurate predicⁿ.
2. The commonly used value of K is 5.
3. Always use odd number as the value of K. ( to avoid ties, prevent bias to specific classes).