Model and Cost function

Not^n's :

m : no. of training ex.s

x's : input var.s/ features

y's : output var.s/ ~~features~~ target var. that we are trying to predict

(x, y) : one training ex. / observ^n

$(x^{(i)}, y^{(i)})$: i^{th} training ex. / observ^n

$(x^{(i)}, y^{(i)})$; $i = 1, 2, \ldots, m$: training set

i : index into the training set

X : space of I/P val.s

Y : space of O/P val.s

n : no. of features

$$\boxed{\text{Hypothesis} : h_\theta(x) = \theta_0 + \theta_1 x}$$

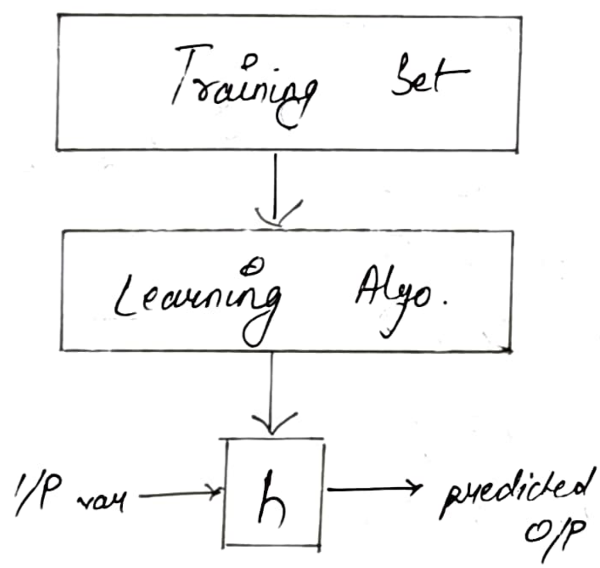
θ_i's : para. of model

h maps from x's to y's

In supervised learning, our goal is : given a training set to learn a fun^n $h: x \rightarrow y$ so that $h(x)$ is a "good" predictor for the corresponding val. of y

How it works : we find a training set (ex: Housing price size in sq. ft on x vs \$ on y) into our learning algo. The learning algo. then outputs a fun^n 'h' called hypothesis (terminology)

This fun^n h takes I/P val and predict the desired O/P (here, takes feet^2 & predict price)

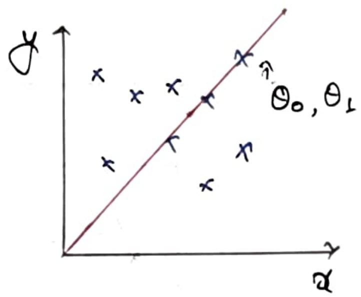


→ When the target var. that we are trying to achieve is continuous, such as in our housing problem, we call the learning probl. a Regression probl.

→ When you can take only a small no. of discrete val.s (such as, if given the living area, we wanted to predict if the dwelling is a house or an apartment), we call it a Classification problem.

Cost Function

let us figure out how to fit the best possible straight line to our data



Idea : choose θ_0, θ_1 so that $h_\theta(x)$ is close to 'y' for our training ex.s (x, y)

i.e., $h_\theta(x) - y$ is min for all training set (diff. b/w hypoth. & ~~a~~ actual o/p)

θ_0, θ_1 (θ_i's) are para. of model. we need to find the ~~closest~~ val. of para.s closest to the actual val.s. For this, we will calc. diff b/w our predicted val. & og val for each data set. For that we use **Cost Fun**

We can measure the accuracy of our hypothesis fun^n by using a cost function.

This takes an avg. difference (actually a fancier ver of an avg) for all results of the hypothesis w/ i/ps from x's and actual o/p, y's.

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

$$\underset{\theta_0, \theta_1}{\text{Minimize}} \; \underbrace{J(\theta_0, \theta_1)}_{\text{cost fun}^n}$$

$$h_\theta(x_i) = \theta_0 + \theta_1 \cdot x^{(i)}$$

To break it apart, it is $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of squares of $h_\theta(x_i) - y_i$, or the diff. b/w the predicted val. & the actual val.

This fun^n is otherwise called the "squared error fun" or "mean squared error".

The fun^n is halved ($\frac{1}{2}$) as a convenience for the comput^n of gradient descent, as the derivative term of the sq. fun^n will cancel out the $\frac{1}{2}$ term.

Cost Function - Intuition I

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make a straight line (defined by $h_\theta(x)$) wh. passes through these scattered pts.

Our <u>objective</u> is to get the <u>best possible line</u>.

The best possible line would be such so that the avg. sq.ed vertical distances of the scattered pt.s from the line will be the least.

ideally, the line should pass through all the pt.s in our data set. In such a case, $J(\theta_0, \theta_1) = 0$.

Now mathematically speaking, each val. of θ_1 corresponds to a diff. straight line fit or a diff. ~~ex~~ hypoth ($h_\theta(x)$) and for each val. of θ_0, θ_1 you can derive ~~some~~ some val. of $J(\theta_0, \theta_1)$.

So ~~we~~ for visualizing what cost fun^n does & why we use it, we calc. val.s of $J(\theta_0, \theta_1)$ for diff. val. of hypoth (i.e.; $h_\theta(x) = \theta_0 + \theta_1 x$) and plot them in a graph. We then look at the graph and figure out for what val. of θ_0, θ_1 is $J(\theta_0, \theta_1)$ lowest i.e; we $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$, wh. is our goal.

ex: let, $\theta_0 = 0$

so, $h_\theta(x) = \theta_0 + \theta_1 x \Rightarrow h_\theta(x) = \theta_1 x$

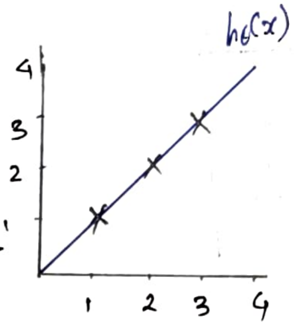
Now,

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

(for fixed θ_1, it is a fun^n of x)

so graph of this fun^n will be the slope $h_\theta(x)$ plotted in x/y axes

for $\theta_1 = 1$



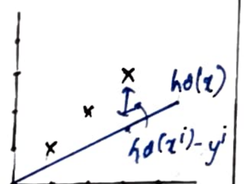
$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

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

$= \frac{1}{2\times 3} \left((0-0)^2 + (0-0)^2 + (0-0)^2\right)$

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

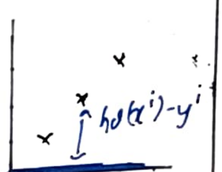
for $\theta_1 = 0.5$



$J(\theta_1) = \frac{1}{2\times 3} \left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$

$J(0.5) = 0.68$

for $\theta_1 = 0$



$J(0) = \frac{1}{2\times 3}(1^2 + 2^2 + 3^2)$

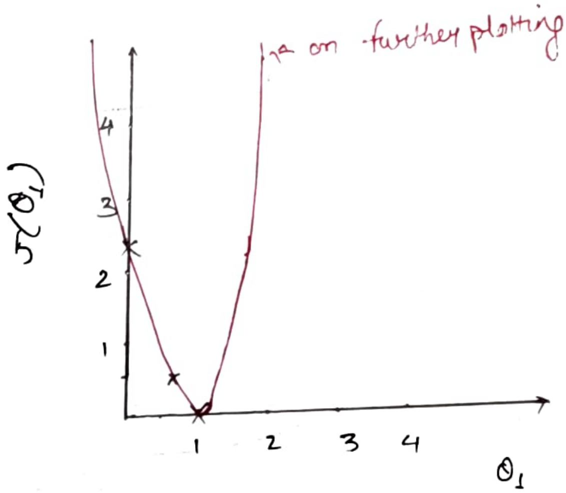
$J(0) = \frac{14}{6} = 2.3$

$J(\theta_1)$

(fun^n of para θ_1)

so the graph of this fun^n will be plotted in $\theta_1 / J(\theta_1)$ axes (only when $\theta_0 = 0$)

→ plotting $J(1)$, $J(0.5)$, $J(0)$ in $J(\theta_1)/\theta_1$ axes (only bc $\theta_0 = 0$)



Clearly, $J(\theta_1)$ is min at when $\theta_1 = 1$.

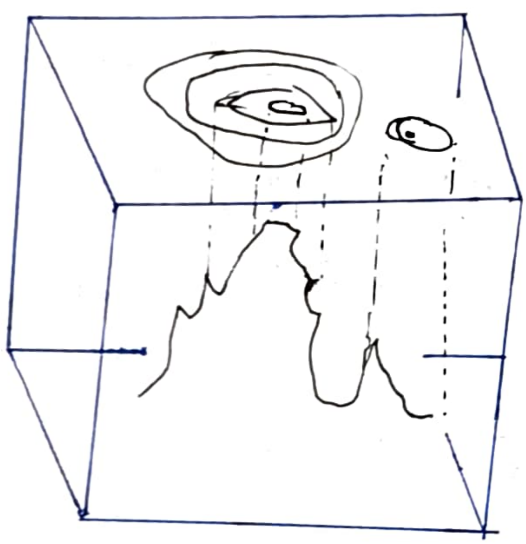
Objective completed.

Cost function - Intuition II

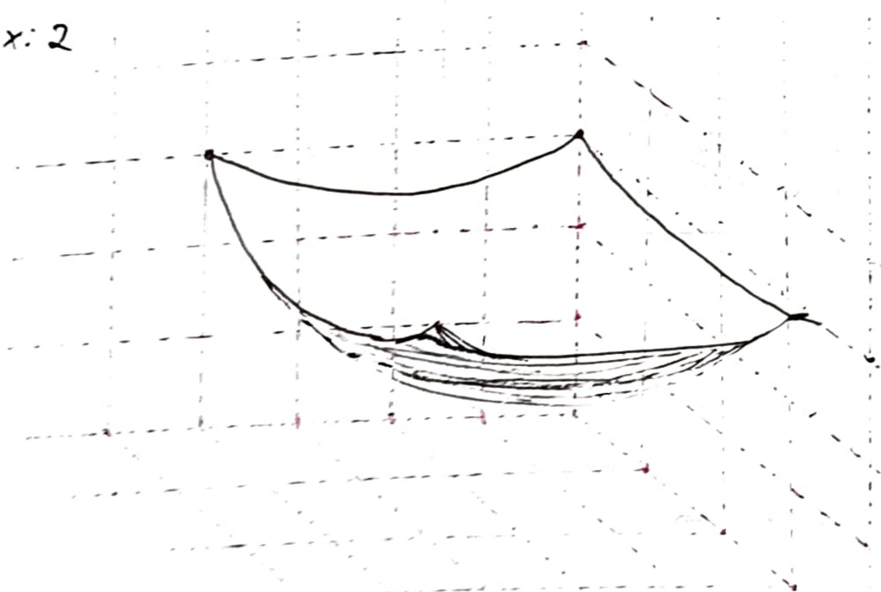
a Contour plot is a graph that contains many contour lines. A contour line of a 2 var. fun has a const. val. at all pt.s of the same line.

a Contour plot is a graphical technique for plotting a 3-D surface by plotting const. z slices called 'contours' on a 2-D format.

ex: (1)



ex: 2



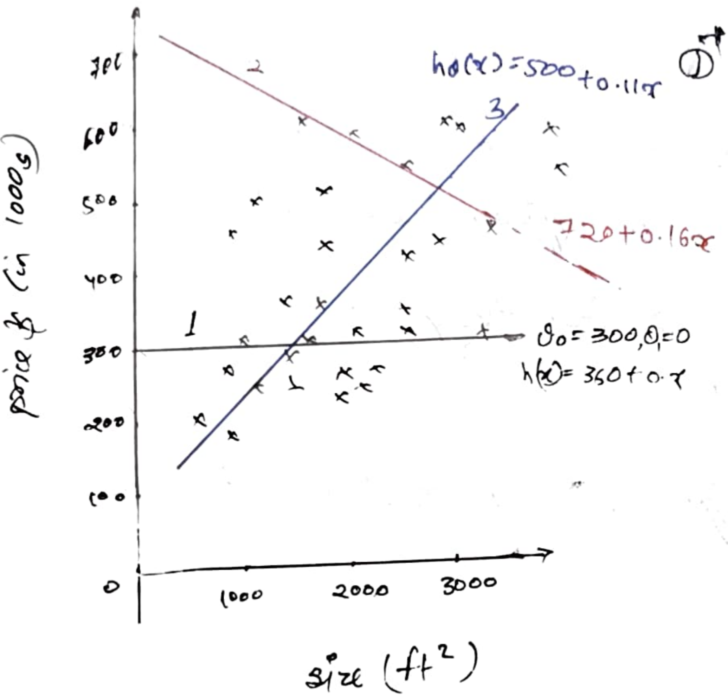
3-D [illegible] is drawn in the following table

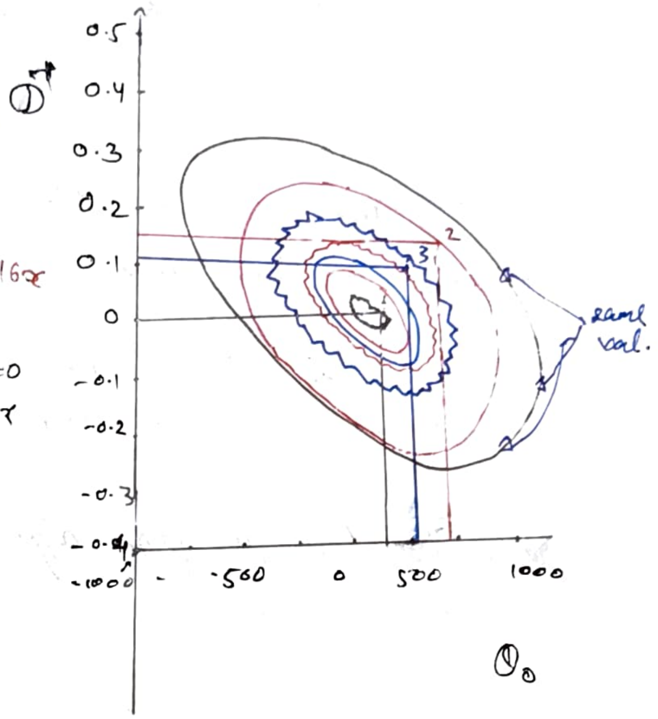
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a fun^n of x)

$J(\theta_0, \theta_1)$

(fun^n of para.s θ_0, θ_1)





1 $(\theta_0 = 300, \theta_1 = 0)$

2 $(\theta_0 = 720, \theta_1 = 0.16)$

3 $(\theta_0 = 500, \theta_1 = 0.11)$

all pts. on the same contour line have same val. so, all 3 Δs have equal val (same val. of $J(\theta_0, \theta_1)$)

~~~~~ are linear lines like — or / only. They are drawn such only bc I have 3 colour pens.

Gradient Descent for Linear Regression

here we will put together grad. desc. w our cost fun^n and that will give us an algo for Linear Regression, or putting a straight line to our data.

grad. desc. algo :

repeat until convergence {

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

(for $j = 0$ & $j = 1$)

}

Linear Regression Model :

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

Linear hypoth.

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

sq.ed error cost fun

Now, we need to calc. $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

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

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

so, for

(partial diff. w.r.t θ_j)

$$j=0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right)$$

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

putting them in grad. desc. algo

repeat until convergence {

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

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

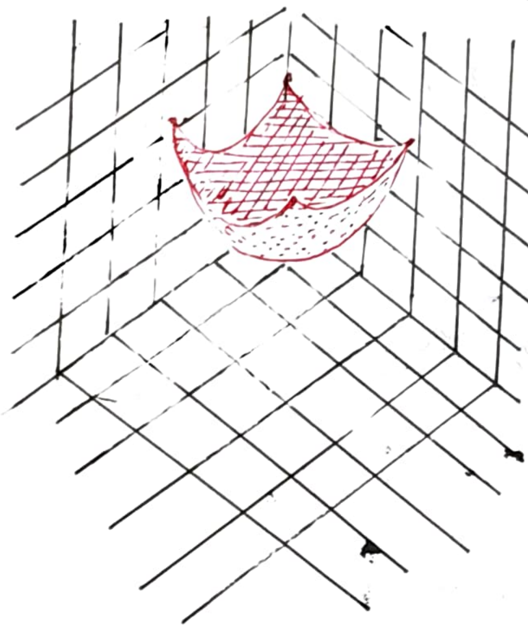
This algo is also k/a Batch grad. desc. (orig) where Batch means that every step of grad. desc. uses all training ex.s $\sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right)$

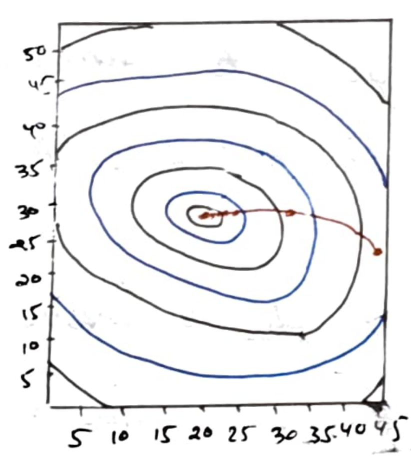
However, some algo.s also use subsets of training ex.s and not all of them.

If we start w a guess for our hypoth. & then repeatedly apply grad. desc. eqⁿs, our hypoth. will become more accurate.

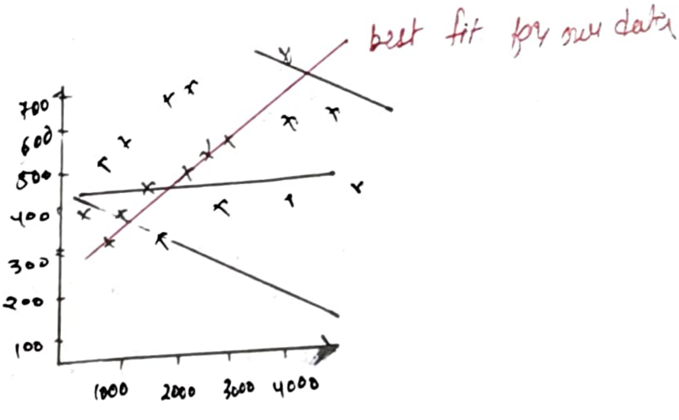
☆ the cost funⁿ for Linear Regression is always going to be a convex funⁿ (bow-shaped funⁿ) and so, it doesn't have the local optimum except the one global optimum.
so, the grad. desc. will always converge to the 1 local optimum.

$J(\theta_0, \theta_1)$ funⁿ of paras θ_0, θ_1





⇓



Multivariate Linear Regression

Linear Regression w multiple var.s is k/a "multivariate linear regression".

→ i.e., we have multiple features / var.s w wh. we try to fit a model to our data.

ex:

size (ft^2) x_1	no. of bedrooms x_2	no. of floors x_3	age of home (yrs) 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
				⋮

47

We now introduce nota'n for eq's where we can have any no. of i/p var.s

m = no. of training ex.s $= 47$

n = no. of features $= 4$

$x^{(i)}$ = i/p (features) of i^{th} training ex. $\quad x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$

↳ i → index to training set

$x_j^{(i)}$ = val. of feature j in i^{th} training ex. $\quad x_3^{(2)} = 2$

$x_1^{(4)} = 852$

similar to our hypo. for linear regr. $h_\theta(x) = \theta_0 + \theta_1(x)$

The multivariate form of the hypothesis function accomodating these multiple features is as follows:

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

★ for convenience, we assume $x_0^{(i)} = 1 \ \forall (i \in 1, .., m)$ in this course.

This allows vec.s 'θ' & $x^{(i)}$ match each other element wise ($n+1$ el.s)

i.e., $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ & already $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$

Now,

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

$$= \begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix}_{1 \times n+1} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n+1 \times 1}$$

$$y = \boxed{h_\theta(x) = \theta^T x}$$

Gradient Descent for Multiple Variables

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

Para.s : $\theta_0, \theta_1, \theta_2, \ldots, \theta_n = \theta$

$\theta = [\theta_0, \ldots, \theta_n]$ is an $n+1$ dimension vec.

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

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

Gradient Descent

for single feature / var,
for $n = 1$

Repeat

{

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

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

$\hookrightarrow x_j^{(i)}$

(simultaneously update θ_0, θ_1)

}

The gradient descent equa^{n} itself is generally the same form, we just have to repeat it for our 'n' features.

$n \geq 1$.

repeat until convergence

{

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

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

}

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

sim. to prev θ_0 (for $n=1$) as for convenience, we have assumed $x_0 = 1$

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

sim to prev θ_1 (for $n=1$)

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

Features and Polynomial Regression.

We can improve our features and the form of our hypoth. fun^n in a couple diff. ways.

We combine multiple features into one.

ex: Housing Price predic^n:

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

Instead of using the features we have in hand, we may create another feature wh. helps us create a better model.

$$\text{Area } x = \text{frontage} \times \text{depth}$$

then, $h_\theta(x) = \theta_0 + \theta_1 x$ ← land area

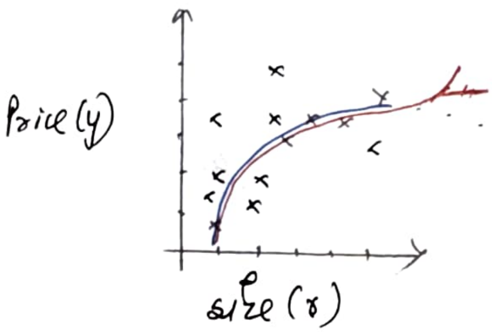
Polynomial Regression → how to put polynomial (quadratic, cubic) fun^n into data).

Polynomial Regression allows us to use the machinery of Linear Regression to fit very complicated, even non-linear data

Our hypo. funⁿ need not be a linear (straight line) if it does not fit the data well.

We can change the behaviour or curve of our hypo funⁿ by making it quadratic, cubic or sq. root funⁿ (or any other form).

if you have data set



you can fit a model

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

← ($\because$ quad. funⁿ's ~~are~~ go up then come down)

or, $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

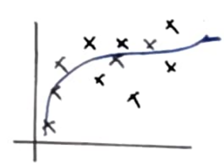
(sim. reason : cubic eqⁿ goes up)

$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$$

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

or

$$= \theta_0 + \theta_1 (size) + \theta_2 (\sqrt{size})$$



:

When fitting such model to our data, feature scaling becomes extremely imp. bc

ex. $x_1 = size = 1 - 1000\ ft^2$

$x_2 = size^2 = 1 - 10^6\ ft^2$

$x_3 = size^3 = 1 - 10^9\ ft^2$.

Normal Equation

Gradient Descent gives 1 way of minimizing J. Let's discuss a second way of doing so, this time performing the minimization explicitly & w/o resorting to an iterative algo.

In the "Normal Equation" method, we will minimize J by explicitly taking its (partial) derivatives w.r.t the θ_j's, and setting them to 0.
This allows us to find the optimum θ w/o iter^n.
The normal eq^n formula is given below:

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

→ Intu^n: If 1D ($\theta \in \mathbb{R}$)

to minimize $J(\theta)$

$$\frac{d}{d\theta} J(\theta) \xrightarrow[\text{set}]{} 0$$

solve for θ (gives us optimum val of θ)

but in our problem, $\theta \in \mathbb{R}^{n+1}$ (θ is n+1 dimensional vec.)

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

so, $\frac{\partial}{\partial \theta_j} J(\theta) \overset{set}{=} 0$ (for every j)

solve for $\theta_0, \theta_1, \ldots, \theta_m$ — gives us the vals of θ that minimize the cost fun^n $J(\theta)$.

→ Examples :

$m = 4$

we added this col. x_0	size (ft^2) x_1	no. of bedrooms x_2	no. of floors x_3	age of home (yrs) 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

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}_{m \times (n+1)} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}_{m \times 1}$$

then,

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

gives us the vals of θ that minimizes the cost fun^n.

→ In gen.,

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

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \quad \& \quad \underset{\substack{\uparrow \\ \text{k/a} \\ \text{design matr.}}}{X} = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

ex: if $x^{(i)} = \begin{bmatrix} 1 \\ x_2^{(i)} \end{bmatrix} \Rightarrow X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix}$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

then, $\theta = \cancel{X^T} (X^T X)^{-1} X^T y$.

→ There is no need to do feature scaling w the Normal Eqⁿ.

~~for~~ more complex algo.s like Classificⁿ and Logistic Regression cannot be solved using Normal Equation. for them we have to use grad. desc.

→ Octave: $\boxed{\text{pinv}(x' * x) * x' * y}$

→ Comparison b/w Gradient Descent & Normal Eq^n

for m training ex.s, n features.

Grad Desc.	Normal Eq^n
→ ~~do~~ need to choose α	→ no need to choose α
→ needs many iter^n's	→ no need to iterate
→ $O(kn^2)$	→ $\{\Theta\alpha\ O(n^3)$, causes need to calc. $(X^TX)^{-1}$
→ works well when n is large ↳ $n > 10^6$	→ slow if n is very large.

Normal Equation and non-invertibility.

If X^TX is non-invertible (singular / degenerate), the common causes might be having

- redundant features (linearly dependent) ex: $x_1 = ft^2$, $x_2 = m^2$
- too many features (ex: $m \leq n$) In this case, del some features or use "regulariz^n".

in MATLAB/OCTAVE, use pinv() instead of inv() to calc $(X^TX)^{-1}$

Regularized Linear Regression.

Optimiz^n objective:

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

(regularized lin regr

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

Grad. Desc.:

Repeat

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

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

The term $\frac{\lambda}{m} \cdot \theta_j$ performs our regulariz^n.

Update rule on further manipul^n:

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

$\because m > 0, \alpha > 0 \Rightarrow 1 - \frac{\alpha\lambda}{m} < 1$ (ex 0.99)

What the above expr. means is that $\forall\ j = 1, \ldots, n$, for every iter^n we are updating θ_j a little by first regularizing (shrinking) it a little - $\left(1 - \frac{\alpha\lambda}{m}\right)$, and then performing a similar (grad. desc.) update as before $\left(-\alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}\right)$.

It is just intu^n, mathematically it is just performing grad. desc. on regularized cost fun^n.

Normal Eq^n:

To add in regulariz^n, the eq^n is same as og, except that we add another term inside parantheses.

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}_{m \times (n+1)} \text{ mat}$$

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{m \times 1} \text{ vec.}$$

$$\boxed{\begin{aligned} \theta &= (X^T X + \lambda \cdot L)^{-1} X^T y \\ \text{where } L &= \begin{bmatrix} 0 & & 0 \\ & 1 & \\ 0 & & \ddots \\ & & & 1 \end{bmatrix}_{(n+1) \times (n+1)} \end{aligned}}$$

ex: $n = 2$

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{3 \times 3}$$

in regularized lin. regr., if λ is set to an extremely large val., algo results in Underfitting. bc $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$.

if $\lambda = 10^4$ (say) $\Rightarrow \theta_1, \theta_2, \ldots, \theta_n \approx 0 \Rightarrow h_\theta(x) = \theta_0$