UNIT 1 : Linear Classifier and Generalizations

LECTURE 4 : Linear classification and Generalizations

Wooooooooo !!!!!!! We are back baby !!!!! Yeahhhhhh !!!!!

After the successful completion of Fundamentals of Statistics , your pal **Scooby** is back with a brand new set of handwritten lecture notes. I found that many of you liked them and appriciated the work by thanking me personally on my email. Which by the way is :

Scooby's email : **peepeepoopoo873@gmail.com**

Feel free to point out mistakes and message me anyways

For those who do not message me...
I don't care man!!! Whataever man !!!! Screw you guys !!!! Whatever!!!

(I do care, please message me. I was just pretending to be indifferent in the above line so that you can't hurt me emotionally.) Anyways.

I tried to make these notes without any mistakes and cover most of the video material.

But as Yoda has rightly said : ' *Is human, To err* '

I however **DO NOT TAKE** any responsibilty if you fail your exams following my notes. So please check the actual lecture slides and videos if you suspect something is wrong, and mail me too.

The notes **DO NOT CONTAIN** solutions to any homework problems or in-lecture exercises. Only the problems and exercises solved in the videos are included in these notes.

Here is an interesting photo I googled

HAPPY SEAL

I ran out of made up quotes so I will write very short stories instead. These may or may not resemble true events, but will definitely have a message that will help you along the journey, that is life.

STORY TIME :

So an astronaught was roaming Mars in his fancy suit, when he came across an alien looking thingy. To his suprise the alien knew his language so they started chatting.

The astronaut asked the alien if he lived in Mars and if there were any other lifeforms like him on Mars. The alien told the astronaut that he indeed was from a different planet, and just like him he was also an astronaut.

Our astronaut was shocked. That meant there were other worlds with space exploration capabilities, with other life forms. So they both talked a lot about life in their respective planets and their lives there, and the technologies of their planets.

As they talked more, the human asked the alien how they reproduced. The alien told the human they have this thing called 'gond' in between there legs, which is used to impregnate alien chicks in their planet. The human was shocked, it was the exact same procedure on planet earth.

They talked a lot, had some space beers and spent the night on Mars. The alien told the human how he scored a lot of chicks in his planet. The human became jealous. They argued over who was a better playahh. Finally they decided to measure there d**ks in order to prove superiority.

So they stripped off their space suits and got butt naked. But the atmosphere of Mars would not allow any one of them to breathe properly. They both died suffocating.

The human with his dong in his hand and the alien with his gond in his hand lay lifeless on Mars.

THE END

MORAL : No matter which planet dudes belong to. They will always have a d**k measuring competition if their ego is hurt.

I hope these notes will help you

Let's get started

▷ Lambda : λ

In the previous Lecture, we set up our problem and concluded on the note that in order to find our large margin classifier, ie a decision boundary and two equidistant margin boundaries, we need to minimise the objective function given below wrt θ and θ_0

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) + \frac{\lambda}{2} \|\theta\|^2$$

Now we do not know much about this hyperparameter called λ, so we will first study a little about it before we move on to finding solution for this problem. Lets do that

➤ Developing Intuition for λ I : Weighing factor

Let me remind you how the problem was setup. The idea was to fit a broad margin on the data, instead of just a decision boundary. So we came up (actually Vladimir Vapnik came up) with the idea of a decision boundary with two equidistant margin boundaries, that will fit right into our data to classify the points. The idea was to stretch these boundaries, as far as possible, to get a fat/large boundary.

But there was a problem and it was a big one. As we stretched the margins further apart, for a given train data. The points naturally started coming inside the region bw decision & margin boundaries. If this continues for long the whole purpose of these margin boundaries would become obsolete. Because all the points will come inside. So we came up with the idea of penalising the points, if they are misclassified, or inside the fat boundary (made by positive and negative boundaries). The Loss term was written as

$$\text{Loss}_h(z) = \begin{cases} 1 - z & \text{if } z \leq 1 \\ 0 & \text{if } z > 1 \end{cases} \qquad \text{where } z = y^{(i)}(\theta \cdot x^{(i)} + \theta_0)$$

We also had this interpretation for z

$$z = \frac{y^{(i)}(\theta \cdot x^{(i)} + \theta_0)}{\|\theta\|} \Big/ \frac{1}{\|\theta\|} = \frac{\text{Signed distance of ith point from D.B.}}{\text{Distance bw any positive/negative margin and the decision boundary (D.B)}}$$

So the distance of ith point from D.B (signed) = z times Distance of any margin boundary and the D.B

i.e. The distance of ith point from D.B (signed) $= z * \frac{1}{\|\theta\|}$

We are trying to understand the role of λ, so the next thing we do is see how it's used.

We have our loss term ready, & we now want to minimise the Loss on average

$$\frac{1}{n}\sum_{i=1}^{n} Loss_h\left(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\right)$$

We also wanted to maximise the distance bw the margins

maximise $\frac{1}{\|\theta\|}$ equivalent to minimise $\|\theta\|$

equivalent to minimise $\frac{\|\theta\|^2}{2}$

But these two forces are counter to one another. So we will need to minimize a combination of them and thats where λ comes in. λ weighs the proportion that the margin width ie regularization term will have wrt the loss term. Lets elaborate further

Suppose I have a linearly seperable data set for simplicity. The first thing we must note is for a linearly seperable case, a decision boundary can always be found. Ie a decision boundary that will classify <u>ALL</u> the points correctly. (We can do this with perceptron.)

Now note that if that's the case the the Loss for ith point

$Loss_h\left(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\right)$ will always be bounded bw 0 and 1

because $z = y^{(i)}(\theta \cdot x^{(i)} + \theta_0)$ have 3 choices

- → will be > 1 when ith point is outside its respective margin boundary. In that case $Loss_h(z) = 0$
- → will be < 0 when ith point is misclassified, but that can't be the case because we have already assumed that the data is linearly seperable and a decision boundary can be found that classifies all points correctly
- → will be between 0 and 1, when ith point is between the decision boundary and it's respective margin boundary. In that case $Loss_h(z) = 1 - z$ is also between 0 and 1

So for a linearly seperable case, where a decision boundary can be found classifying all the data points correctly we can say that the quantity :

..the quantity $\sum_{i=1}^{n} Loss_h(z_i)$ where $z_i = y^{(i)}(\theta \cdot x^{(i)} + \theta_0)$

will <u>AT MAX</u> be equal to n (ie total points)

So the average loss

$$\frac{1}{n} \sum_{i=1}^{n} Loss_h(z_i)$$

will be <u>AT MAX</u> equal to 1

Now we can start to look at the objective function, and λ

<u>Case 1.</u> $\lambda = 10^{10}$

Here we see the case where λ is extremely large : 10^{10}

$$J(\theta,\theta_0) = \underbrace{\frac{1}{n} \sum_{i=1}^{n} Loss_h(y^{(i)}(\theta \cdot x^{(i)} + \theta_0))}_{\text{I}} + \underbrace{10^{10} \frac{\|\theta\|^2}{2}}_{\text{II}}$$

The 1st part of $J(\theta,\theta_0)$ ie I can at max be 1 no matter what decision boundary and margin boundary is (we are assuming a proper D.B. for linearly seperable data that classifies <u>all</u> points correctly)

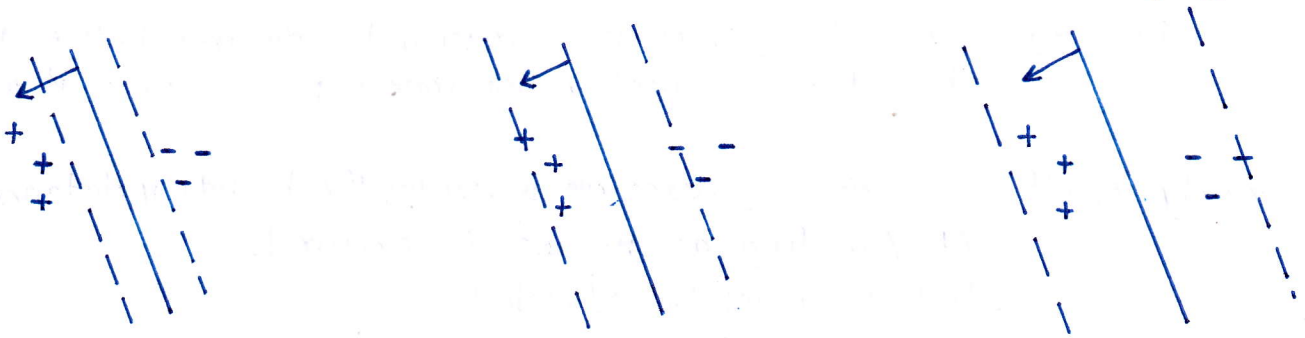
The 2nd part is of the order $10^{10} * \dfrac{1}{\text{distance between any margin \& decision boundary}}$

So the algorithm will obviously prefer to minimise $\|\theta\|^2$ part of the objective function in order to minimise overall $J(\theta,\theta_0)$. ie

$\|\theta\|^2$ will decrease as the algorithm progresses to find solution

$1/\|\theta\|$ will increase as the algorithm progresses to find solution which means

Distance bw the margin boundaries increases as the algorithm progresses

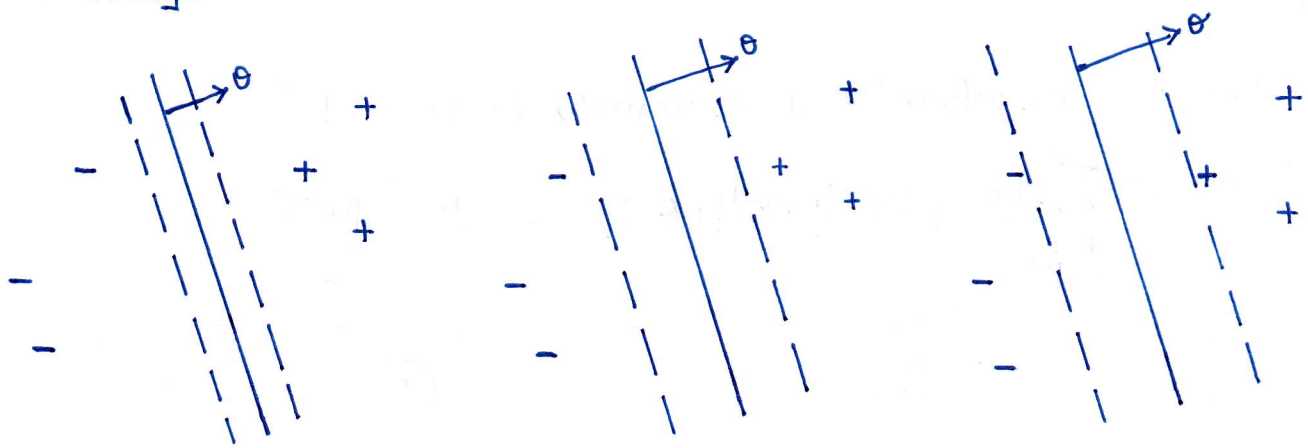


Case 2 : $\lambda = 10^{-10}$

Here we see the case when λ is extremely small : 10^{-10}

$$J(\theta,\theta_0) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \text{Loss}_h\left(y^{(i)}(\theta\cdot x^{(i)}+\theta_0)\right)}_{\text{I}} + \underbrace{10^{-10}\frac{\|\theta\|^2}{2}}_{\text{II}}$$

The first part (I) is of the order of 1, as disussed earlier but the second term (II) is essentially 0 in comparision to (I) because λ is so small. so the algorithm will care only for the 1st term (I) in order to keep $J(\theta,\theta_0)$ as minimum as possible. But this won't happen in a straight forward way.



As long as we do not have any point inside the margin Boundaries, the objective $J(\theta,\theta_0)$ will just be $10^{-10}\frac{\|\theta\|^2}{2}$

So the algorithm will still try to minimize $\|\theta\|^2$ ie stretch the boundaries further apart, but as soon as the 1st point is encountered, the loss term will kick in and if the point enters a little inside the margin say $z = .999$ the loss will be $1 - .999 = .001$ which is still 10^7 times larger compared to 10^{-10} (which is λ's value). So the algorithm cannot afford even a single point to get inside boundaries, so it will stop will points ON the boundary

These were the extreme cases we saw

λ : extremely large : Algorithm focuses on increasing the distance between boundries, no matter how many points come inside

λ : extremely small : Algorithm focuses on increasing the boundary distance ONLY as long as no loss is occured.
Ie Loss cannot be afforded

Now Lets look at the general case

Case 3 : General λ

Now for the case of general λ, we again look at the objective function

$$J(\theta,\theta_0) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} Loss_h(z_i)}_{\text{I}} + \underbrace{\lambda \frac{\|\theta\|^2}{2}}_{\text{II}}$$

we already know that I and II are counterforces. when λ is very small I will be the dominant part, when λ is very large II will be the dominant part.

we also know that I is of the order of 1 units. ie it's max value is 1 and for the normal cases, I will be around 1

So taking this as a reference point, the 2nd term II will be comparable to order 1 when $\|\theta\|^2$ will be of the order $\frac{1}{\lambda}$. because then $\lambda \cdot \frac{\|\theta\|^2}{2}$ will be of order 1.

This gives us the interpretation that when

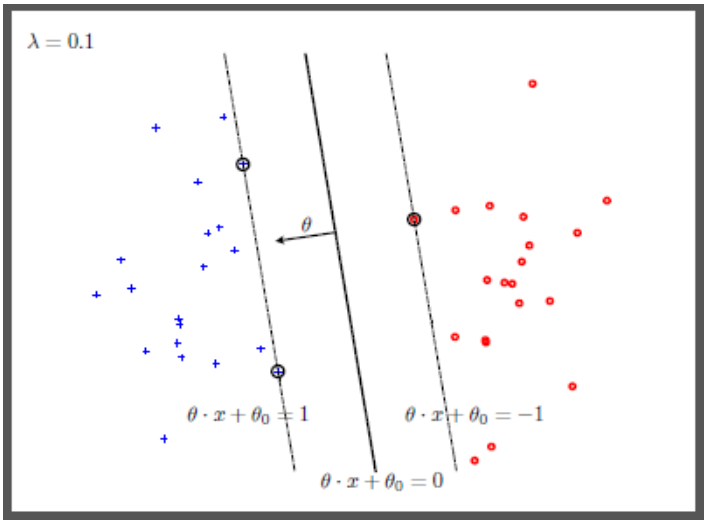
$$\|\theta\|^2 \approx \frac{1}{\lambda} \implies \|\theta\| \approx \frac{1}{\sqrt{\lambda}}$$

or $$\frac{1}{\|\theta\|} \approx \sqrt{\lambda}$$

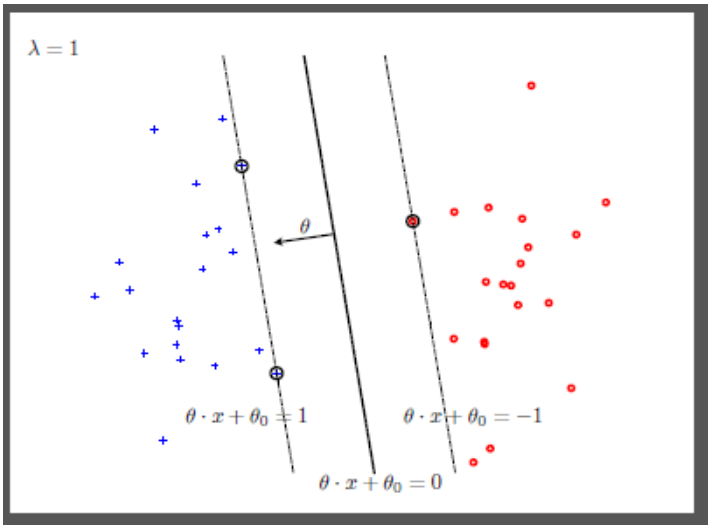
then both terms I and II are in a sort of equal balance, ie they contribute equally to the objective function $J(\theta,\theta_0)$ in terms of their respective quantities to minimise

So $\sqrt{\lambda}$ is the distance b/w any margin & decision boundary when both terms I & II give similiar contribution to $J(\theta,\theta_0)$

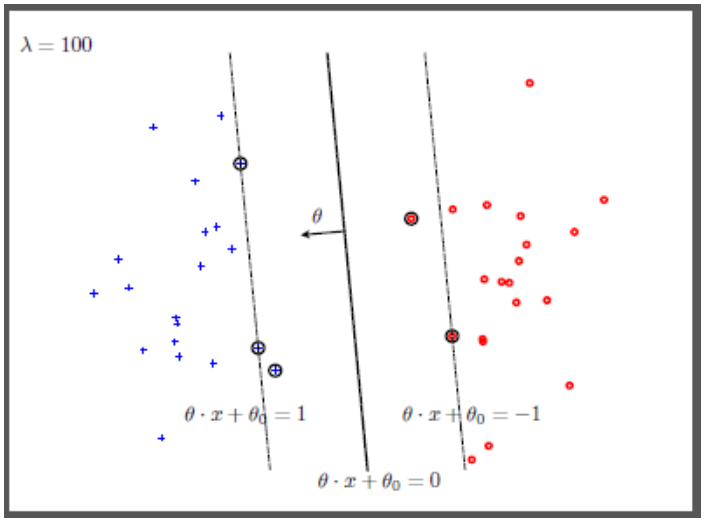
Prof T shows how changing λ changes the fat boundary for a given train dataset. Lets look at the cases of seperable & non-seperable data.



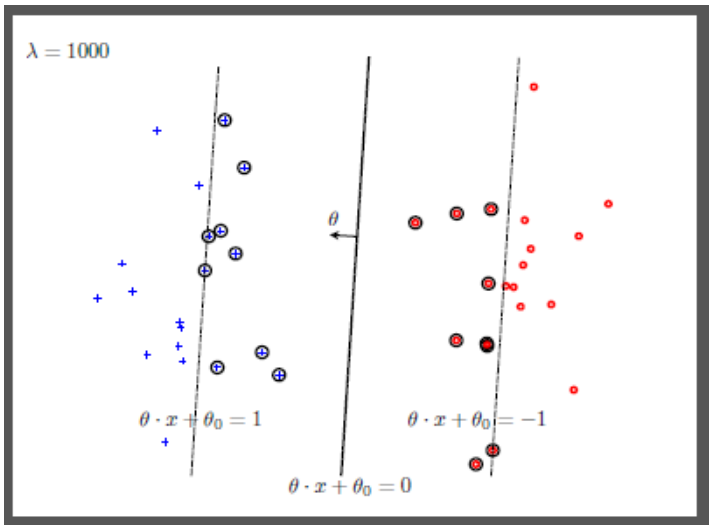
Here we see the lambda is small relative to the loss (of order 1), so the focus is on minimising the loss hence no point is allowed inside the margin boundaries.



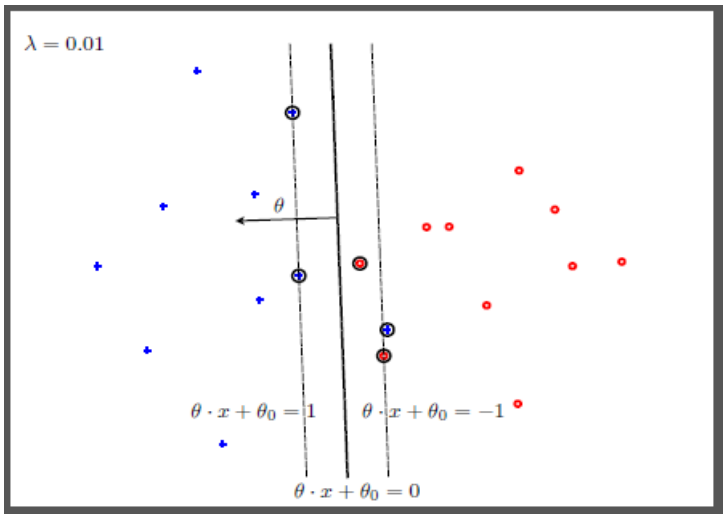
Here we see the lambda is 1 now, but the optimal solution is still the one that does not allow points inside the margin boundaries.

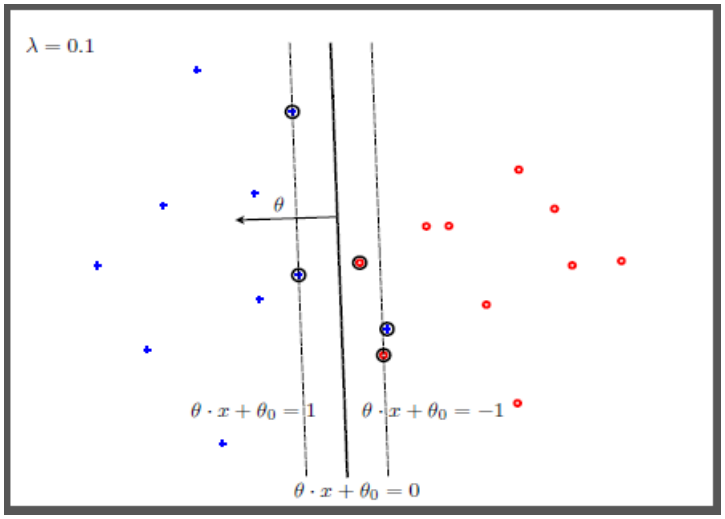


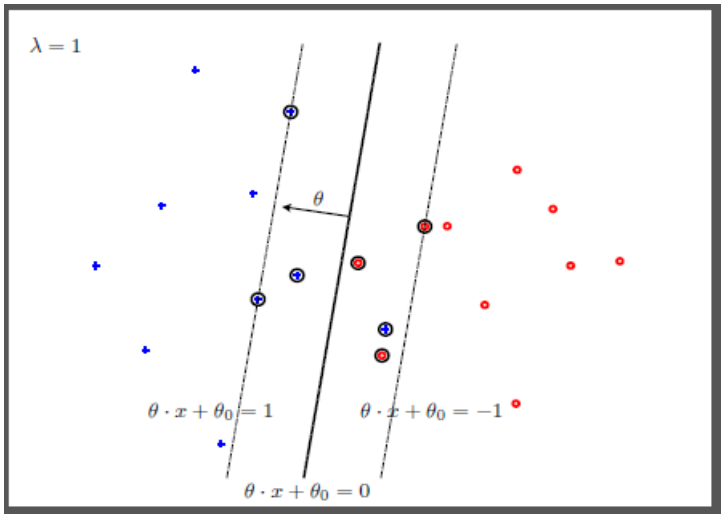
Now when lambda is 100, we can see that the optimal solution goes for a broader boundary, even if some points come inside the margin boundaries.

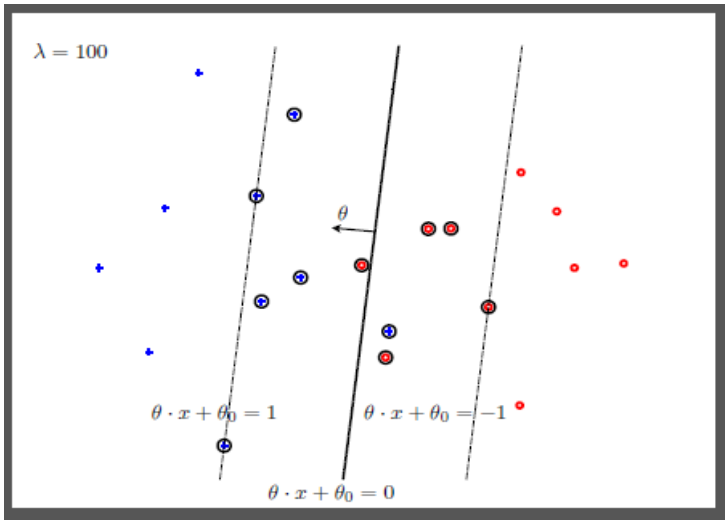


Now when lambda is 1000, which is extremely large relative to the loss term, more and more points are allowed inside the margin boundaries, in order to maximise the seperation.









Do you see the effect of lambda on this dataset you naughty boi !!!!

➤ Developing Intuition for λ II

Lets see another perspective discussed by Prof. T (T stands for Tommi)

We have the equation for objective function

$$J(\theta,\theta_0) = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}\left(y^{(i)}(\theta\cdot x^{(i)}+\theta_0)\right) + \frac{\lambda}{2}\|\theta\|^2$$

Lets write it other way

$$\frac{1}{\lambda}J(\theta,\theta_0) = \frac{1}{\lambda n}\sum_{i=1}^{n} \text{Loss}\left(y^{(i)}(\theta\cdot x^{(i)}+\theta_0)\right) + \frac{\|\theta\|^2}{2}$$

Let's call $\frac{1}{\lambda n} = c$

c stands for complexity and $c \propto \frac{1}{\lambda}$. So what we analyzed earlier for λ can be accordingly analysed keeping c in mind.

First note that $\frac{1}{\lambda}J(\theta,\theta_0)$ for a given dataset (training) will give the same optimal solution for a given λ (or $\frac{1}{cn}$) which we provide it.

So here's the analysis from before

As λ increases $\longrightarrow$ c decreases $\longrightarrow$ A broader boundary is preffered

As λ decreases $\longrightarrow$ c increases $\longrightarrow$ Boundary that minimizes the loss terms is preferred, even though it may not be broad.

Now how do we evaluate the performance of our large margin classifier. It's through the Loss function ie average loss

$$\text{Test Loss} = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}\left(y^{(i)}(\theta\cdot x^{(i)}+\theta_0)\right)$$

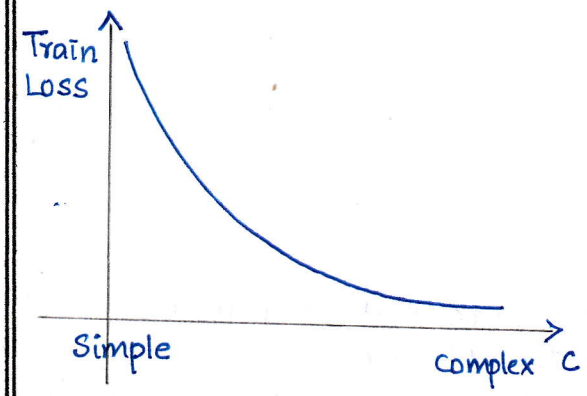
So let's first look at how this loss changes with c for train data

➤ TRAIN DATA (For seperable case. just for simplicity)

- when $c \approx 0$, the loss term in $J(\theta,\theta_0)$ will be zero, so algorithm focuses on decreasing $\|\theta\|^2$ as long as possible. ie width starts increasing. So all the points, even though correctly classified come inside their respective margin boundaries. Therefore the test loss has a large value.
- As c starts to increase, the loss term in $J(\theta,\theta_0)$, starts becoming relevant so some points are thrown out of their respective margin boundaries, ie the width of margin boundaries decrease and they come closer. So the test loss starts decreasing.

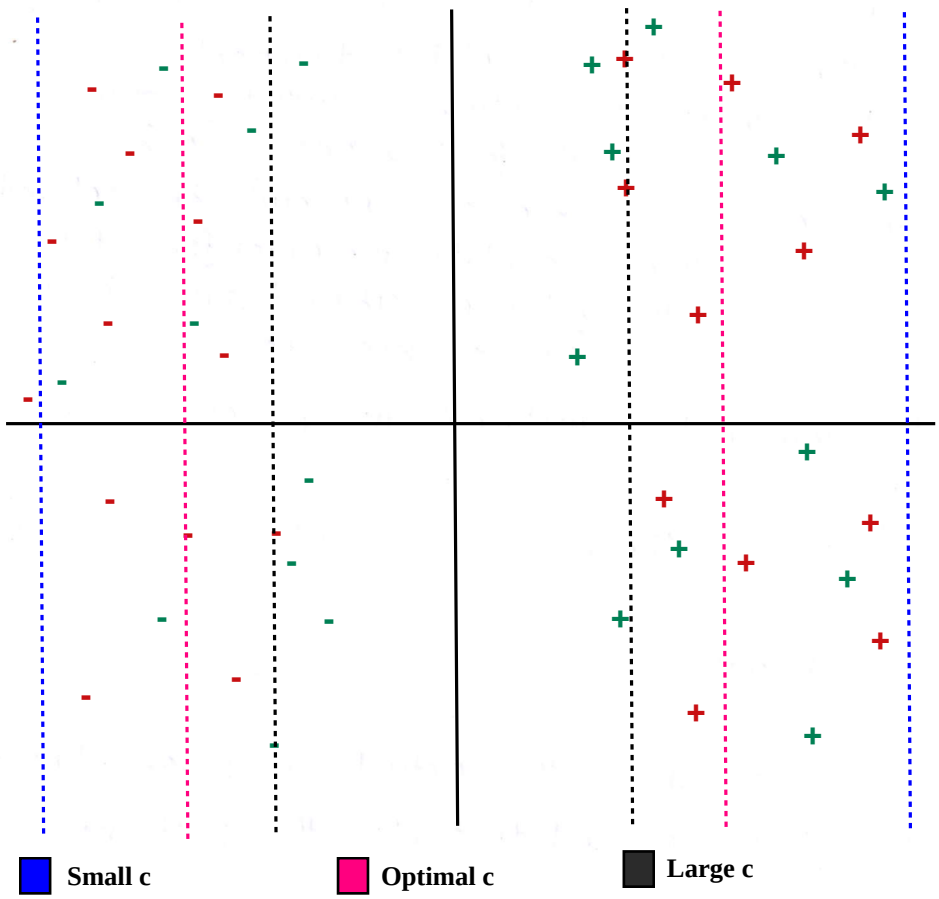
• Now as c becomes extremely Large, the loss term in the $J(\theta, \theta_0)$ is the major contributor, so the algorithm will focus on decreasing it as long as possible. If the data is Linearly seperable then that can be done perfectly, by fitting the fat boundary (decisionboundary + margins) inside the Little empty space between points. In that case the Loss (Test) will be exactly zero. ie the margins will be grazing the nearest points. If the data is not seperable, the loss (Test) will still decrease as c increases but wont go to zero.

so the picture looks like this



Now we see simple & complex written below c axis. What this means is as c increase the solution we get for θ and θ_0 are more and more dependent on the training data. The solution for large c is very specifically formed in order to encapsulate <u>all</u> the points of the training data, and it's very specific for that particular train data, hence its complex, in that sense.

Lets now look at what will happen to Test Loss



when c increases, the loss term in $J(\theta, \theta_0)$ becomes significant & boundaries start to shrink.

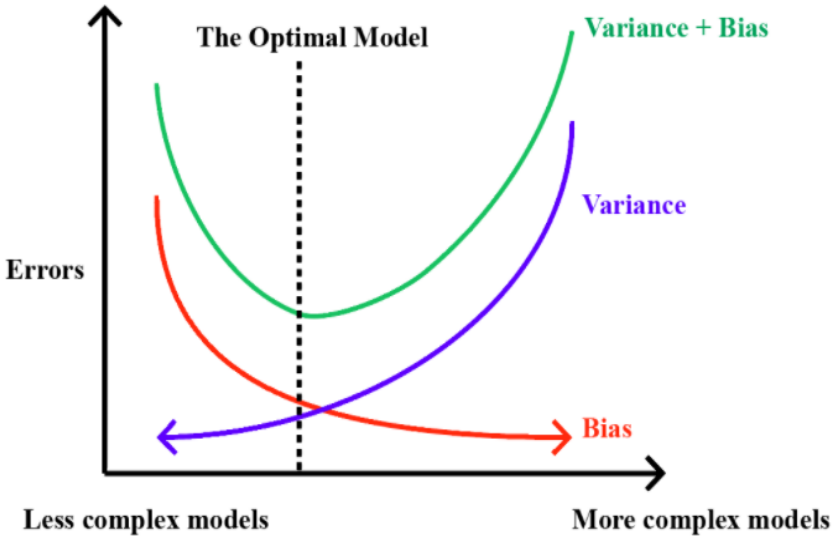
Here the red points are the train data and the green are test data. We are assuming that the test data & train data come from some underlying same process so they will be generated close to each other as shown

Initially as c increases, test error will also increase as the points (test) will go out of the margins.

But as c becomes large there will come an optimal point c^* where test error will be minimum

Then the fit will be an overfit as the classifier will be precisely modified according to large value of c for train data. The model will then be less generalizable for the case of test data

The test error after that point will start to increase.



This relate to bias variance tradeoff which we will study later.

Very low c means we will have a underfitted model
Very high c means we will have a overfitted model

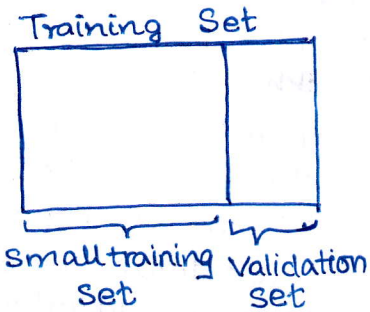
We need to find the optimal c^*

Lets see how we find the optimal c^*

> Cross Validation

How do we find this optimal c^*. or λ^*. We must note that we plug in λ^* or c^* in the $J(\theta, \theta_0)$ before we give it to the computer to optimize

To find the c^* or λ^*, we want a value that minimise the test loss. We can't do that because we do not know the labels of test set beforehand. But what we do instead is we simulate evaluating the test losses, we have our training set (drawn as box)



We divide the actual training set into two parts

1. Smaller training set : Over which we will train our model for a given value of λ
2. Validation set : we think of it as a pseudo test set and we can calculate the error for it by applying the classifier trained above on smaller training set.

We repeat the process for training on the smaller training set and finding validation error for different values of λ or c

We then choose the value that gives the minimum validation error as an estimate for the actual c^* (or λ^*)

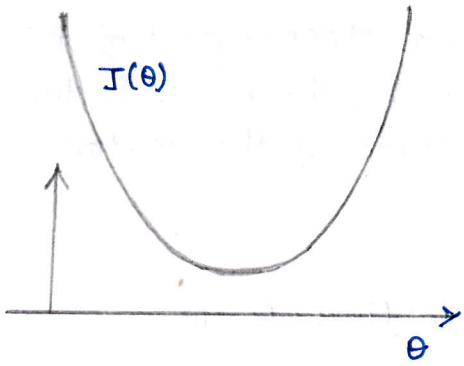
This is the process of cross validation

With that being said, lets now move on to the actual process of optimizing $J(\theta, \theta_0)$. We will see algorithms that are used to tackle this problem

Let's Begin

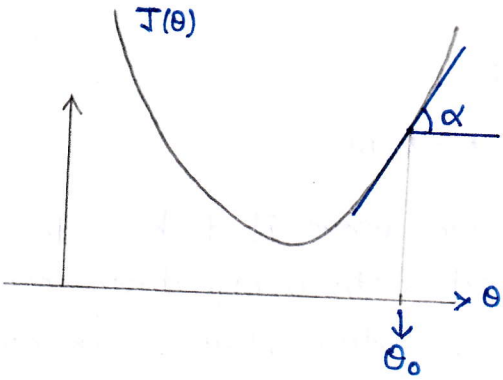
▷ Gradient Descent

Lets start with a simple 1d case. We have a function $J(\theta)$ where $\theta \in \mathbb{R}$ and it's a convex function. So we can draw it as below, just for the sake of visualization. (not necessary that $J(\theta)$ will always look like this)



One thing we know about the derivative is that :

$$\left.\frac{d\,J(\theta)}{d\theta}\right]_{\text{at } \theta=\theta_0}$$ is the slope of the tangent line at point $\theta=\theta_0$



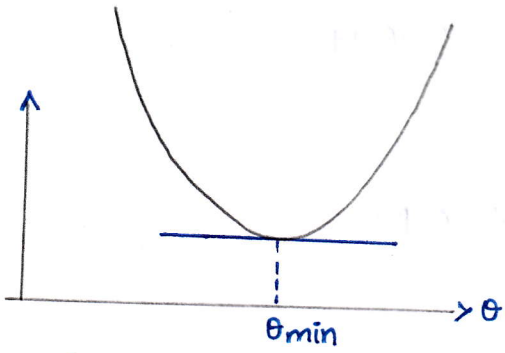
That is to say

$$\tan\alpha = \left.\frac{d\,J(\theta)}{d\theta}\right]_{\text{at } \theta=\theta_0}$$

Now one must note that the magnitude of $\frac{d}{d\theta}J(\theta)$ is more when the curve is steeper and less when curve is not so steep. Infact we already know

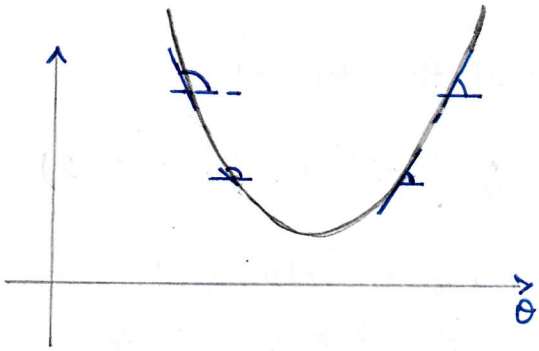
At the minima

$$\left.\frac{dJ(\theta)}{d\theta}\right]_{@\,\theta=\theta_{min}} = 0$$



Now suppose I make a vector in the direction of $\hat{\theta}$ ie "$\longrightarrow$"
$\hat{\theta}$ is the unit vector for direction and the vector I make is

$$-\frac{dJ(\theta)}{d\theta}\,\hat{\theta}$$

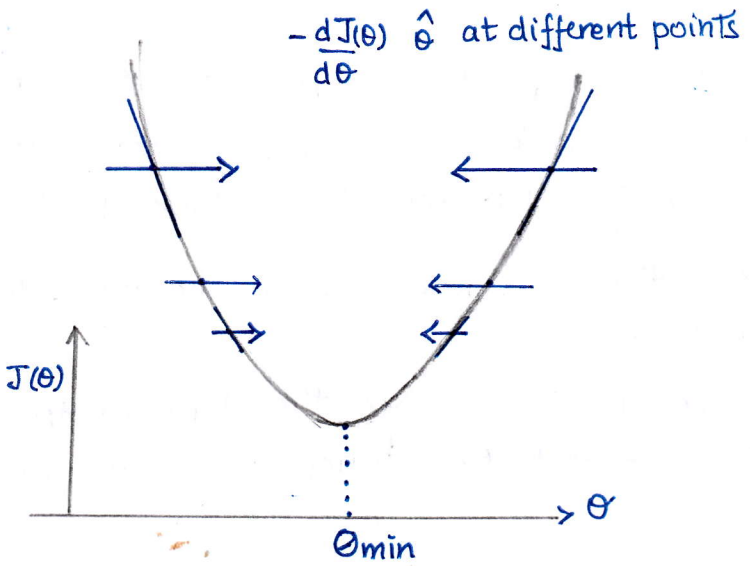


So this vector will have magnitude

$$\left\|-\frac{dJ(\theta)}{d\theta}\hat{\theta}\right\| = \left\|\frac{dJ(\theta)}{d\theta}\right\|\|\hat{\theta}\| = \left\|\frac{dJ(\theta)}{d\theta}\right\|$$

And it will point in the direction of $\hat{\theta}$ when $\frac{dJ(\theta)}{d\theta}$ is negative

and away from direction of $\hat{\theta}$ when $\frac{dJ(\theta)}{d\theta}$ is positive



So we can use this vector $-\frac{dJ}{d\theta}\hat{\theta}$ to observe that it always points towards θ_{min}

The more steeper part of the $J(\theta)$ we are in, the more is the magnitude of this vector

So I can use this fact to find an algorithm to find θ_{min}

Suppose I start from any initial point $\theta = \theta_{init}$ and on each of the iteration, I update as follows

$$\theta_{new} = \theta_{old} - \eta * \left.\frac{dJ(\theta)}{d\theta}\right]_{@\,\theta=\theta_{old}}$$

Here η is called the Learning rate and we want it to be relatively small. Note that, here we are concerned with a one dimensional vector $\theta \in \mathbb{R}$, so we don't use the vector notation. But above equation can equivalently be written as

$$\vec{\theta}_{new} = \vec{\theta}_{old} - \eta * \left[\frac{dJ}{d\theta}\right]_{@\,\theta=\theta_{old}} \hat{\theta}$$

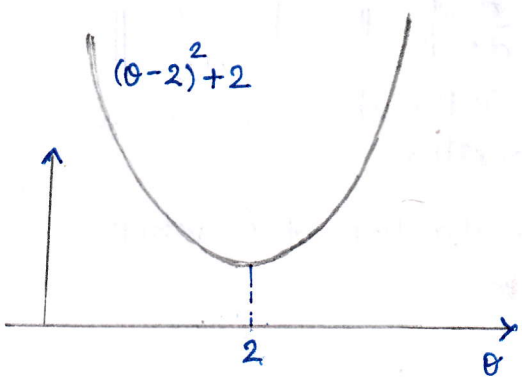
where $\hat{\theta}$ is a unit vector along θ axis

Let's see how this plays out with an example

$\boxed{\text{Eg.}}$ Lets say $J(\theta) = (\theta-2)^2 + 2$

we already know for this example that the minima will be obtained at

$\theta_{min} = 2$



We will need the derivative :

$$\frac{dJ(\theta)}{d\theta} = \frac{d}{d\theta}(\theta-2)^2 + 2 = 2(\theta-2)$$

for a given value of θ

Lets say we start with $\theta_{initial} = 0$ & $\eta = \cdot 1$

Iteration 1 : $\theta_1 = \theta_{init} - \eta * \left.\frac{dJ(\theta)}{d\theta}\right]_{\theta=\theta_{init}}$

$$\theta_1 = 0 - (\cdot 1) * (2(0-2)) = 0 + (\cdot 1) * (4) = \cdot 4$$

Iteration 2 : $\theta_2 = \theta_1 - \eta * \left.\frac{dJ(\theta)}{d\theta}\right]_{\theta=\theta_1}$

$$\theta_2 = \cdot 4 - (\cdot 1) * (2(\cdot 4 - 2)) = \cdot 4 + (\cdot 1) * 3.2 = \cdot 72$$

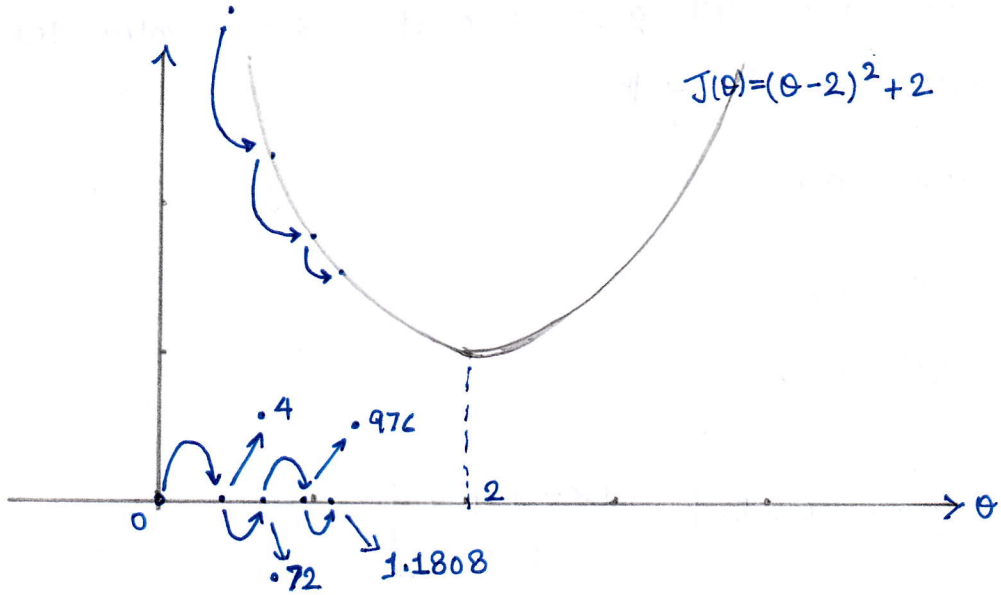
Iteration 3 : $\theta_3 = \theta_2 - \eta * \left.\frac{dJ(\theta)}{d\theta}\right]_{\theta=\theta_2}$

$$\theta_3 = \cdot 72 - (\cdot 1) * (2(\cdot 72 - 2)) = \cdot 72 + \cdot 256 = \cdot 976$$

Iteration 4 : $\theta_4 = \theta_3 - \eta * \left.\frac{dJ(\theta)}{d\theta}\right]_{\theta=\theta_3}$

$$\theta_4 = \cdot 976 - (\cdot 1) * (2(\cdot 976 - 2)) = \cdot 976 + \cdot 2048 = 1.1808$$

And so on we will find the answer for $\theta_{min} = 2$



Note how the leap size decreases as we come closer to θ_{min} because the updation rule / algorithm takes $\frac{dJ(\theta)}{d\theta}$ at any point and substracts it. for far away points $\frac{dJ(\theta)}{d\theta}$ is large due to graph being steep in comparision to points near θ_{min} so leap size is large for far away points

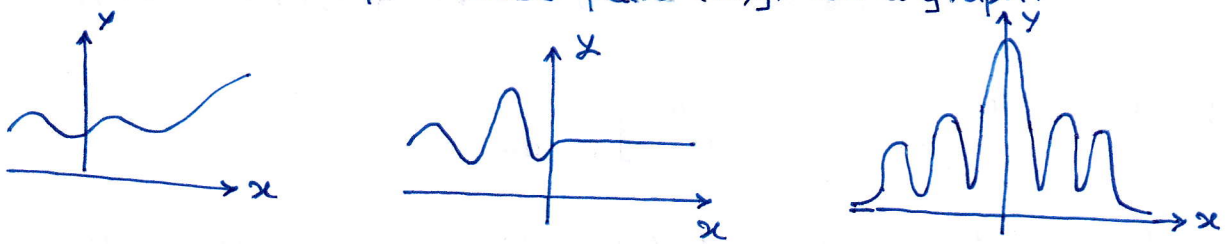
So for the one dimesional case for $J(\theta)$ where $\theta \in \mathbb{R}$, we can find $\theta_{min} \equiv \text{argmin } J(\theta)$ by the rule

$$\theta_{K+1} = \theta_K - \eta * \left.\frac{dJ(\theta)}{d\theta}\right]_{\theta=\theta_K}$$

Lets move on to more than 1 dimensions but, before that we need

MATHEMATICAL INTERLUDE : ∇ Grad

What is a function? It's a mapping (DUH!!)
A function of 1 variable $f(x)$ takes in 1 argument, real value x and and gives us an output $y = f(x)$. For all values of x, we can get output y and we can also plot these pairs (x,y) on a graph.



Similiarly we can increase the dimension of our function. We can decide that our function will take two inputs : x and y.
But the function will still give as output 1 real value denoted by z
So $z = f(x,y) \quad : \mathbb{R}^2 \rightarrow \mathbb{R}$

Some examples are:

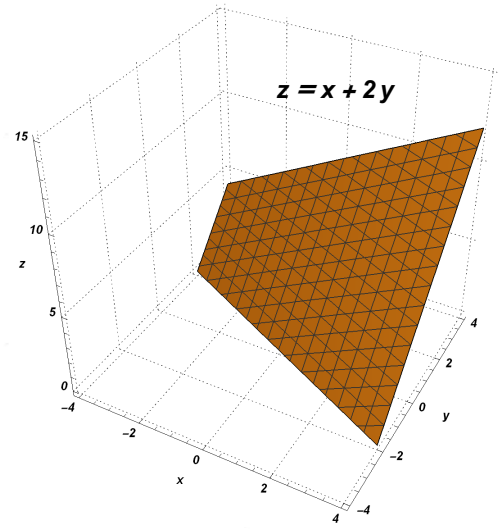
$$f(x,y) = x + 2y$$

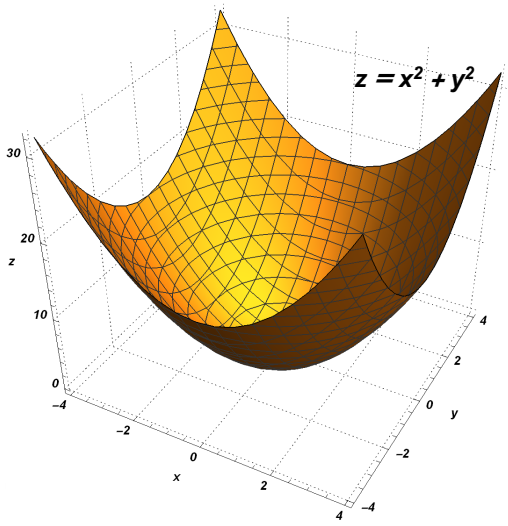
$$f(x,y) = x^2 + y^2$$

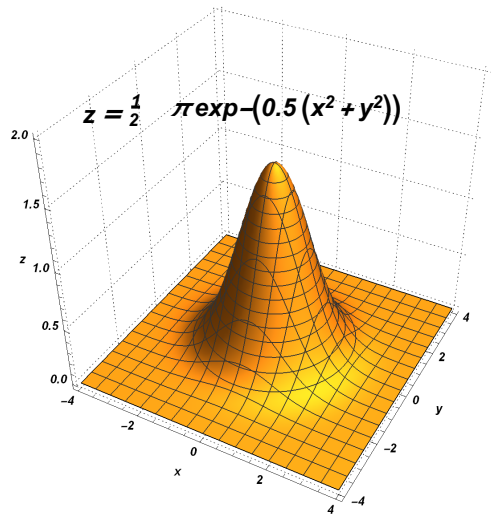
$$f(x,y) = \frac{1}{2\pi} \exp\left[\frac{-(x^2+y^2)}{2} \right]$$

We can similiarly find the value of $z = f(x,y)$ for different values of (x,y) and plot all the points of the triplet (x,y,z) in a 3d space to visualise the graph of $f(x,y)$

That's done next.







Now as in the case of 1 dimensions, we can take derivatives here too. But here we have two real valued inputs ie x and y. So we will have two derivatives, and this is compactly done using the vectors/algebra notation. We take derivative of $f(x,y)$ wrt x and y and stack those up in a column. This is called the gradient of $f(x,y)$

$$\text{Gradient of } f(x,y) \equiv \vec{\nabla} f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) \\ \frac{\partial}{\partial y} f(x,y) \end{bmatrix}$$

Note the gradient is a vector that will be different for each point (x,y)

Lets try to find them for some examples

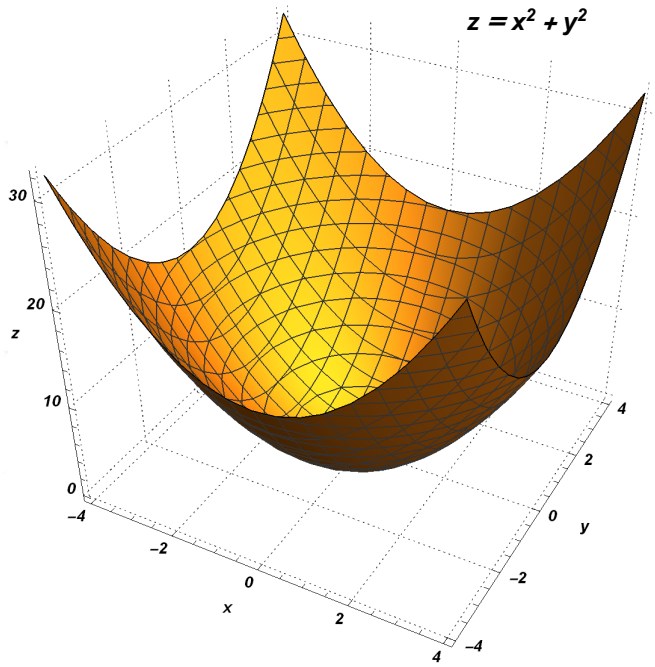
$$|1\rangle \quad f(x,y) = x + 2y \quad ; \quad \vec{\nabla} f(x,y) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

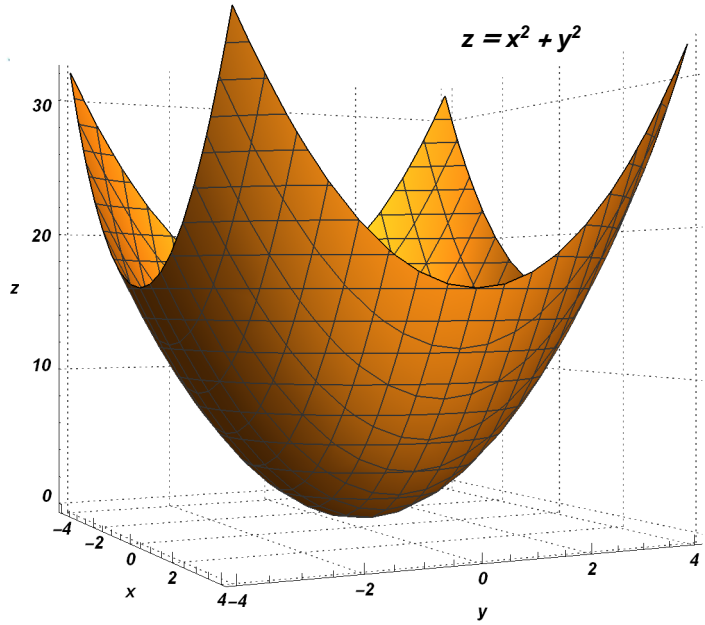
$$|2\rangle \quad f(x,y) = x^2 + y^2 \quad ; \quad \vec{\nabla} f(x,y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

$$|3\rangle \quad f(x,y) = \frac{1}{2\pi} \exp\left[\frac{-(x^2+y^2)}{2}\right] \quad ; \quad \vec{\nabla} f(x,y) = \begin{bmatrix} -x \cdot \frac{1}{2\pi} \exp\left[\frac{-(x^2+y^2)}{2}\right] \\ -y \cdot \frac{1}{2\pi} \exp\left[\frac{-(x^2+y^2)}{2}\right] \end{bmatrix}$$

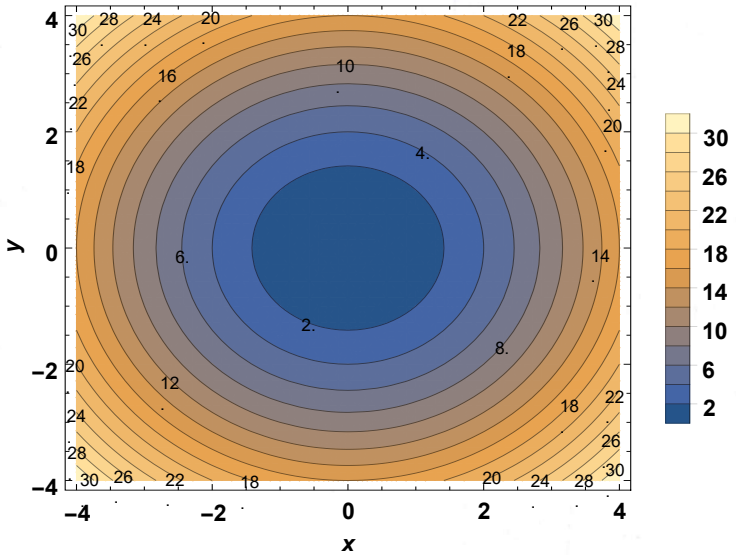
If $\vec{\nabla} f(x,y)$ is a vector then it must point some where

Lets see where it points for the 2nd example $f(x,y) = x^2 + y^2$





Here I have plotted the function $f(x,y) = x^2+y^2$. We will see where the $\vec{\nabla} f(x,y) = \begin{bmatrix} x \\ y \end{bmatrix}$. But before doing that note that this is a vector which does not depend on z. so it will be same for all values of z once (x,y) pair is fixed. So we can see the top view of the above surface to see how $\vec{\nabla} f(x,y)$ looks for $f(x,y) = x^2+y^2$. To show that we will in fact plot $-\vec{\nabla} f$ as that's the vector of interest to us. But first the look of $f(x,y)$ from above



In order to see how $f(x,y)$ x^2+y^2 looks from above, we plot what are called contour plots each curve represents a fixed value for function $f(x,y)$

For eg

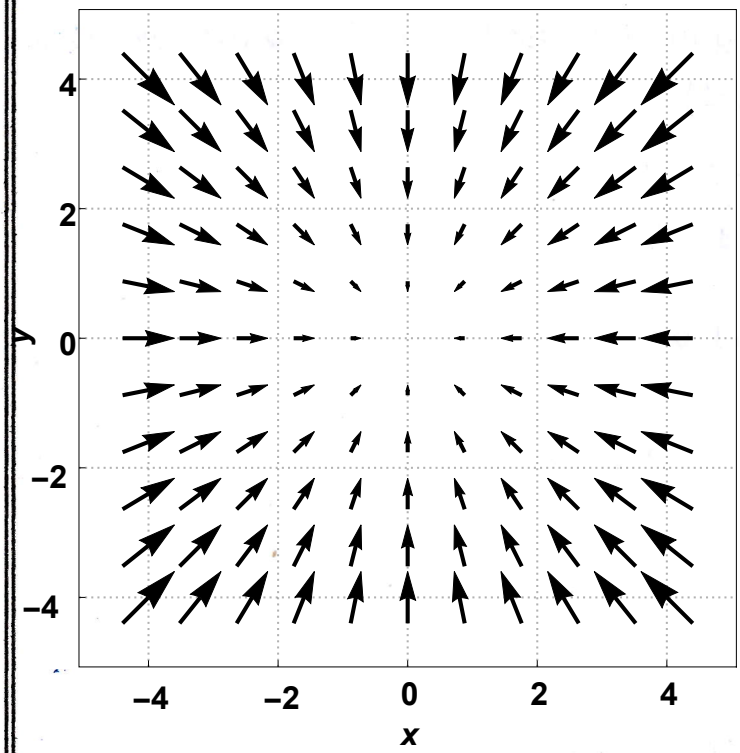
$f(x,y) = 2$ is

$x^2+y^2 = 2$ the equation of a circle which is Labelled in the graph. as 2

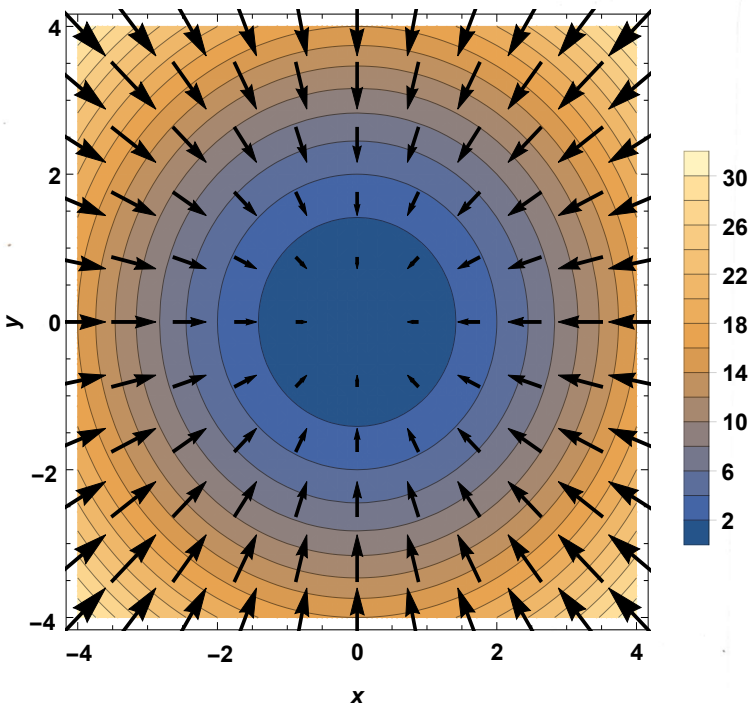
You can think of these curves as the curves we get when the 3d plot of $f(x,y)$ is cut by a plane $z =$ constant, we get the curve

$f(x,y) = z =$ constant $= x^2+y^2$

Next we see what happens when we plot the vectors $-\vec{\nabla} f(x,y) = \begin{bmatrix} -x \\ -y \end{bmatrix}$ for different values of (x,y). Again we don't have any dependence on z. so lets do it & visualize it on x-y plane.



These are how the vectors $-\vec{\nabla} f = \begin{bmatrix} -x \\ -y \end{bmatrix}$ look for different values of x and y. They clearly point towards $(0,0)$. This will be more clear when we superimpose, the vectors over the contour plots, drawn earlier.



Here we see what's really happening. The vectors at any point points towards contours (also called level curves) of lesser value $f(x,y)$ then the current value on that particular point.

And this is not a coincidence. for a function $f(x,y)$. The gradient $\vec{\nabla} f(x,y)$ has the direction has the direction in which function $f(x,y)$ increases most quickly as compared to all the other directions.

The magnitude of gradient is the rate of fastest increase for $f(x,y)$

we are here analysing $-\vec{\nabla} f(x,y)$ so it will give us the direction and rate of fastest decrease. We can use this property of gradient to find an update rule for $J(\vec{\theta},\theta_0)$ and we will do that, to find $\vec{\theta}_{min}$

But before that, let me show how this interplay plays out for another example. I will take the function

$$f(x,y) = \exp\left[-\left\{\frac{(x-1)^2+(y-1)^2}{2}\right\}\right] - \exp\left[-\left\{\frac{(x+1)^2+(y+1)^2}{2}\right\}\right]$$

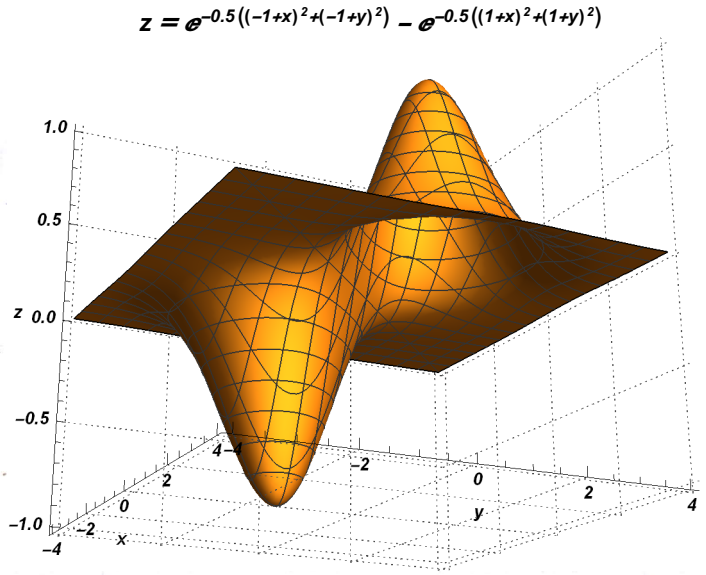
Let's see how that plays out

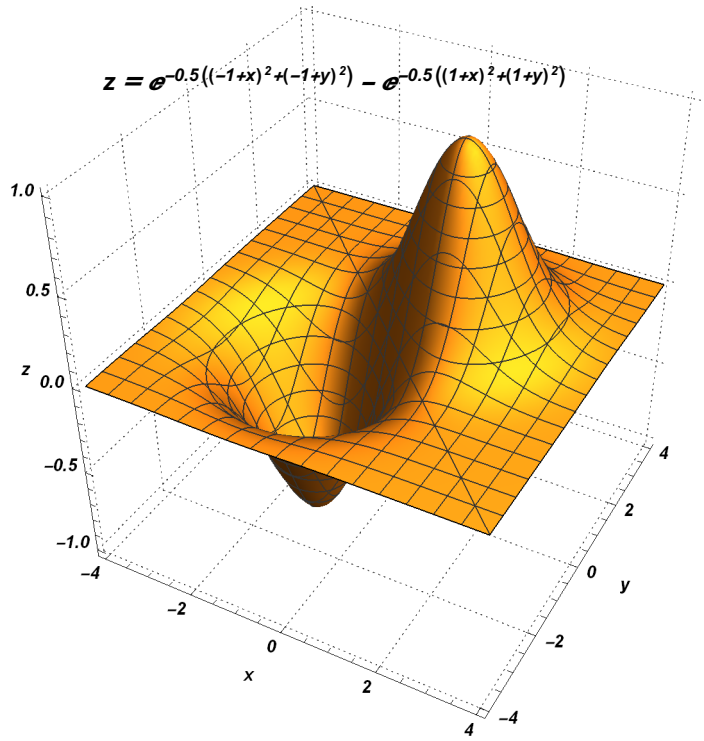
The function is essentially a sum of gaussian and negative gaussian, one centered at (1,1) and other at (-1,-1).

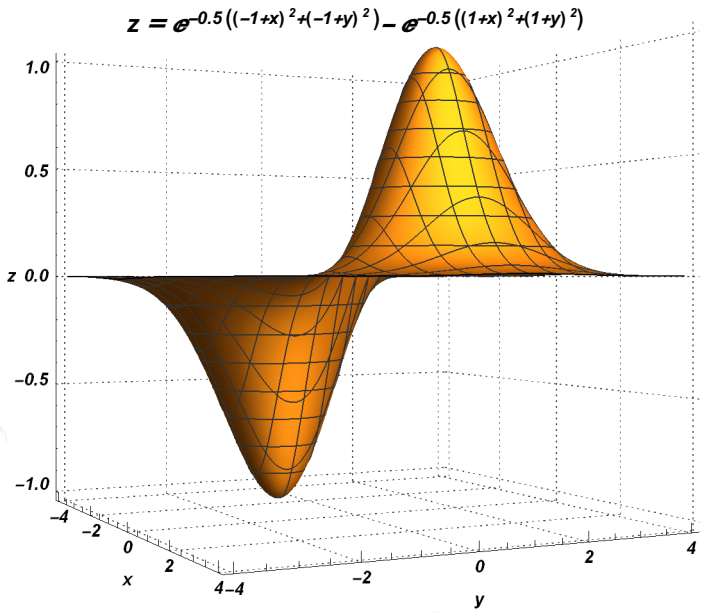
The minima of function is at (-1,-1)

The maxima of function is at (1, 1)

The graph of function looks like below

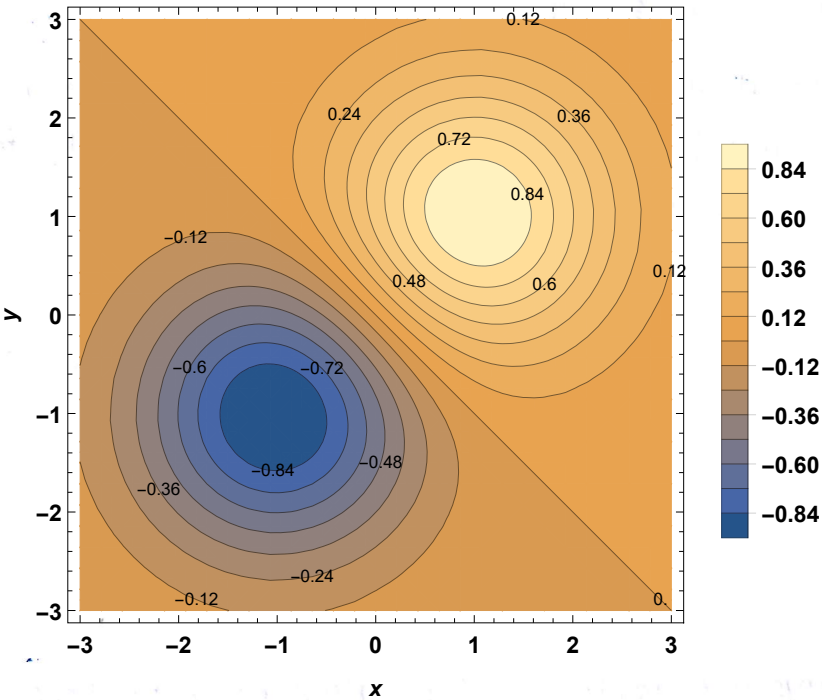


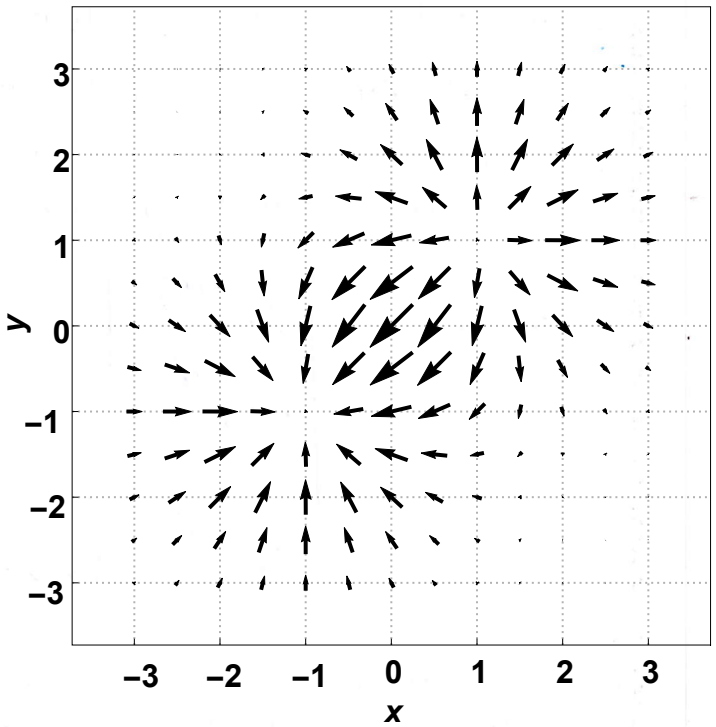


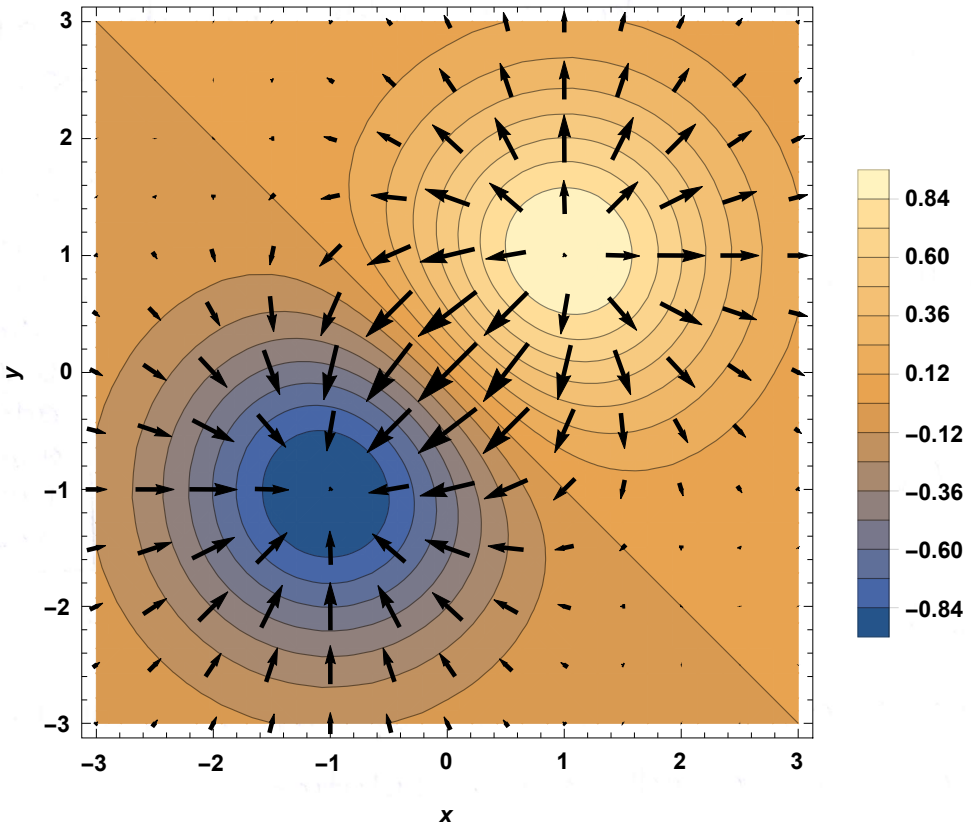


Next we will see, how the grad of $f(x,y)$ and the contours of this function looks like. We will also superimpose these two to see where $-\vec{\nabla} f(x,y)$ points.

I will not calculate $-\vec{\nabla} f(x,y)$ but Mathematica® does it, so it's correct.







We clearly see $-\vec{\nabla} f(x,y)$ points towards the contours of lower value (th_n the current point it's calculated on). Finally if we follow the direction of $-\vec{\nabla} f(x,y)$ we will reach the minima $(-1,-1)$.

Now lets look at the general case of a d dimensional function.

> General Case and Gradient Descent Finally

So similiarly for a d dimensional function $J(\theta)$ where $\theta \in \mathbb{R}^d$, we have the input $\vec{\theta}$ which is a d dimensional vector, but the output is still $\mathbb{R}$, ie a single real valued number. We have

$$\vec{\nabla} J(\theta) = \begin{bmatrix} \frac{\partial}{\partial \theta_1} J(\vec{\theta}) \\ \frac{\partial}{\partial \theta_2} J(\vec{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_d} J(\vec{\theta}) \end{bmatrix}_{d \times 1}$$

In the course we will not always use vector '→' notation over vectors but, its assumed that we know where we use vectors and where scalars

The geometry which may not be visualizable (Yeah, that's a word), but it still remains the same. So we can have our algorithm using $\vec{\nabla} J(\theta)$.

The algorithm will we

> start with some $\vec{\theta} = \vec{\theta}_{initial}$

> Iterate over

$$\boxed{k\text{th step}: \quad \vec{\theta}_{k+1} = \vec{\theta}_k - \eta * \vec{\nabla} J(\vec{\theta})}$$

Note

we can add another element to $\vec{\theta}$ ie θ_0, in order to find the additional θ_0 and the algorithm will remain the same

here we will have a vector equation as $\vec{\theta}$ is a vector.

If the function is convex, then for sufficiently small step size / or Learning rate η, we will find a solution

But this method is algorithmically extensive and let me show you why is that.

Remember we earlier had

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) + \frac{\lambda}{2} \|\theta\|^2$$

Usually, nowadays we have large dataset so $n = 100$ million is not a very big number. Similiarly we can have $d = 100$ features so in each step of the update for gradient descent

$$\theta_{new} = \theta_{old} - \eta * \vec{\nabla} J(\theta)$$

Inorder to compute the above, we need to compute

$$\vec{\nabla} J(\theta) = \vec{\nabla} \left\{ \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(z_i) + \frac{\lambda}{2} \|\theta\|^2 \right\}$$

where $z_i = y^{(i)} (\theta \cdot x^{(i)})$

So

$\vec{\nabla}$ Loss (z_i) → will have d derivatives to be calculated

And this calculation will have to be done for n terms in $J(\theta)$

So for each update we will have to calculate $d \times n$ calculations

So for our case we will have 100million * 100

= 10 Billion calculations

for a single update

So we will next see a better algorithm called Stochastic Gradient Descent that is far less computationally intensive.

The discussion that follow is inspired by a lecture called Stochastic Gradient Descent from the course MIT 18.065 given by Prof. Suvrit Sra. Check it out on youtube.

Let's begin

▷ Stochastic Gradient Descent

S.G.D is a pretty ancient technique that is widely used today to solve large scale machine learning problems today. We basically aim to solve the general problem of the form

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} f_i(\theta) \qquad \text{———} \quad (1)$$

Here θ can be a vector, that's no problem

Remember our objective function

$$\frac{1}{n} \sum_{i=1}^{n} \text{Loss}\left(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\right) + \frac{\lambda}{2} \|\theta\|^2$$

we can take the summation common as the 2nd part does not depend on i so we have

$$\sum_{i=1}^{n} \frac{\lambda}{2} \|\theta\|^2 = \frac{\lambda}{2} \|\theta\|^2 \sum_{i=1}^{n} (1) = n \frac{\lambda}{2} \|\theta\|^2$$

So we can write the objective as

$$\frac{1}{n} \sum_{i=1}^{n} \left[\text{Loss}\left(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\right) + \frac{\lambda}{2} \|\theta\|^2 \right]$$

this is then of the form of equation (1) above where

$$f_i(\theta) = \text{Loss}\left(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)\right) + \frac{\lambda}{2} \|\theta\|^2$$

We can now proceed to write the term inside the square bracket as $J_i(\theta)$, ie the contribution of ith point to the objective function. we can say that the overall objective function is average of simpler objective functions $J_i(\theta)$

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} J_i(\theta) \qquad \text{OR} \qquad J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} J_i(\theta, \theta_0)$$

we will skip θ_0 for simplicity and proceed with 1st form

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} J_i(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left[\text{Loss}\left(y^{(i)}(\theta \cdot x^{(i)})\right) + \frac{\lambda}{2} \|\theta\|^2 \right]$$

So as discussed earlier computing θ_{min} computationally from the update rule

$$\theta_{k+1} = \theta_k - \eta * \nabla J(\theta)$$

can be computationally extensive and each step / update has to do

$$n \times d$$

No of data points forming $J(\theta) = \frac{1}{n} \sum_i J_i(\theta)$

Each feature vector has d entries meaning d partial derivatives needs to be calculated

that can range to trillions per iteration, for large datasets

Instead S.G.D suggest we do the following

At the kth iteration : Pick a random integer from

$$i(k) = \{1, 2, 3, \ldots\ldots n\}$$

Perform the update : $\theta_{k+1} = \theta_k - \eta_k * \nabla J_i(\theta)$

ie just pick randomly one of the n points for training dataset and take the gradient of $\text{Loss}(y^{(i)}(\theta \cdot x^{(i)})) + \frac{\lambda}{2}\|\theta\|^2$ ONLY.

Then use this gradient to update θ, instead of the whole objective function's gradient $\nabla J(\theta)$

One must also note that the learning rate is not fixed like before in gradient descent but rather indexed by k, ie its dependent on the iteration we are in and will change it's value after each iteration, why that is we will see later.

But first see that the number of calculations for each iteration have drastically decreased as $\nabla J_i(\theta)$ calculates only d partial derivatives for the ith random observation. One iteration is n times faster.

But why should this absurd idea work?

This idea is inspired by a very common theme that we saw in the statistics course. For those of you normies that did not do the statistics course, here's the idea by Prof. Rigollet's work

"Most of the statistics is replacing expectation by averages"

Let me elaborate. Look at the following thingy

$$\nabla J(\theta) = \nabla \frac{1}{n} \sum_{i=1}^{n} J_i(\theta)$$

$$\nabla J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla J_i(\theta) \longrightarrow \text{Ⓐ}$$

The idea is simple. Think of the random pair (x, y) being generated from a random process. What this means in probability language is $(x^{(i)}, y^{(i)})$ is generated from a probability density function. And each of the n samples we have in the training dataset are actually n realizations from this pdf. This may or may not be true, but this is what statistical modelling is.

Now

$$\nabla J_i(\theta) = \nabla\left[\text{Loss}\left(Y^{(i)}(\theta \cdot x^{i})\right) + \frac{\lambda}{2}\|\theta\|^2\right]$$

will itself be random because it depends on $x^{(i)}, y^{(i)}$.

When we take the Expectation of this, Which looks something like

$$E[\ \ ] = \int_{\mathbb{R}^d \times Y} \nabla\left(\text{Loss}(y(\theta \cdot x)) + \frac{\lambda}{2}\|\theta\|^2\right) * \text{pdf of } X, Y * dx\, dy$$

Just for an idea I have written this. Don't take it seriously.

This expectation is equal to $\nabla J(\theta)$ is what SGD assumes ie,

So $\nabla J(\theta) = E[g(\theta)]$

where $g(\theta) = \nabla\left[\text{Loss}(Y(\theta \cdot X)) + \frac{\lambda}{2}\|\theta\|^2\right]$

if you compare this to equation Ⓐ above

we are just replacing the expectation by averages

$$\nabla J(\theta) = E[g(\theta)] = \frac{1}{n}\sum_{i=1}^{n} g_i(\theta) \equiv \frac{1}{n}\sum_{i=1}^{n} \nabla J_i(\theta)$$

Here $g_i(\theta)$ and $\nabla J_i(\theta)$ are same, I used g instead to avoid confusion

$$g_i(\theta) \equiv \nabla J_i(\theta) = \nabla\left(\text{Loss}\left(y^{(i)}(\theta \cdot x^{(i)})\right) + \frac{\lambda}{2}\|\theta\|^2\right)$$

ie. SGD assumes

$$\mathbb{E}[g(\theta)] = \nabla J(\theta)$$

ie <u>on average</u> $g_i(\theta) = \nabla\left[\text{Loss}(y^{(i)}(\theta \cdot x^{(i)})) + \frac{\lambda}{2}\|\theta\|^2\right]$ will converge to $\nabla J(\theta)$ for sufficiently Large n

Therefore then we are able to use $\nabla J_i(\theta)$ to update each step because, on average its like using the whole $\nabla J(\theta)$

Statistician say that $\nabla J_i(\theta)$ is an unbaised estimator of $\nabla J(\theta)$

and that why SGD works. Now Lets look at the case of η_k

> η_k : learning rate

As we talked earlier the Learning rate η is now dependent on the iteration we are in i.e, k

we want η_k to satisfy following conditions

① $$\sum_{k=1}^{\infty} \eta_k = \infty$$

② $$\sum_{k=1}^{\infty} \eta_k^2 < \infty \quad \text{(ie finite)}$$

① Allows us to nudge parameters wherever we are towards the minimum (whatever that means)

② we need to reduce the variance from the stochasticity that we are introducing, so we need this

$$\eta_k = \frac{1}{1+k}$$ satisfies the above two constraints for example.

Lets look at some calculations for gradient in our formulation and find how this $\nabla J_i(\theta)$ term will look like

We want to calculate $\nabla J_i(\theta)$ we know

$$J_i(\theta) = \text{Loss}\left(y^{(i)}(\theta \cdot x^{(i)})\right) + \frac{\lambda}{2}\|\theta\|^2$$

$$= \begin{cases} 1 - y^{(i)}(\theta \cdot x^{(i)}) + \frac{\lambda}{2}\|\theta\|^2 & \text{when } y^{(i)}(\theta \cdot x^{(i)}) \leq 1 \\ 0 + \frac{\lambda}{2}\|\theta\|^2 & \text{when } y^{(i)}(\theta \cdot x^{(i)}) > 1 \end{cases}$$

Some calculations we will need here are as follows

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{bmatrix}_{d \times 1} \quad x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}$$

So $\theta \cdot x^{(i)} = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots \theta_d x_d^{(i)}$

& $\|\theta\|^2 = \theta_1^2 + \theta_2^2 + \dots + \theta_d^2$

$$\nabla y^{(i)}(\theta \cdot x^{(i)}) = y^{(i)} \begin{bmatrix} \frac{\partial}{\partial \theta_1} \theta \cdot x^{(i)} \\ \frac{\partial}{\partial \theta_2} \theta \cdot x^{(i)} \\ \vdots \\ \frac{\partial}{\partial \theta_d} \theta \cdot x^{(i)} \end{bmatrix} = y^{(i)} \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix} = y^{(i)} x^{(i)}$$

$$\nabla \|\theta\|^2 = \begin{bmatrix} \frac{\partial}{\partial \theta_1} \|\theta\|^2 \\ \frac{\partial}{\partial \theta_2} \|\theta\|^2 \\ \vdots \\ \frac{\partial}{\partial \theta_d} \|\theta\|^2 \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ 2\theta_2 \\ \vdots \\ 2\theta_d \end{bmatrix} = 2\theta$$

Therefore we will have

$$\nabla J_i(\theta) = \begin{cases} -y^{(i)} x^{(i)} + \lambda\theta & \text{when } y^{(i)}(\theta \cdot x^{(i)}) \leq 1 \\ \lambda\theta & \text{when } y^{(i)}(\theta \cdot x^{(i)}) > 1 \end{cases}$$

Note that unlike the perceptron the SGD will always update θ in every iteration because the condition $y^{(i)}(\theta \cdot x^{(i)}) \leq 1$ only changes the $\nabla J_i(\theta)$ but it is non zero for all cases

So we have finally the SGD that looks like

Stochastic gradient descent

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left[\text{Loss}_h(y^{(i)}\theta \cdot x^{(i)}) + \frac{\lambda}{2}\|\theta\|^2\right]$$

Select $i \in \{1, \ldots, n\}$ at random

$$\theta \leftarrow \theta - \eta_t \nabla_\theta \left[\text{Loss}_h(y^{(i)}\theta \cdot x^{(i)}) + \frac{\lambda}{2}\|\theta\|^2\right]$$

▷ Quadratic Programming

Now if we have a linearly seperable case, then we are sure that we can find a decision boundary that can seperate <u>ALL</u> points properly.

If we do not allow any error (ie loss) which mathematically means :

$$y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1 \quad \forall\ i = 1, 2, 3 \ldots n$$

Then $J(\theta)$ will simply be :

$$J(\theta) = \frac{\lambda}{2}\|\theta\|^2$$

& minimizing this is equivalent to minimising $\frac{\|\theta\|^2}{2}$

This particular form of problem

Find θ, θ_0 st

minimize $\frac{1}{2}\|\theta\|^2$ subject to

$$y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1, \quad i = 1, 2, 3 \ldots n$$

is now falling under a special type of mathematical optimization problem called Quadratic Programming, that is a well studied field and solutions (numerical) can be found by many algorithms

Check out : **the link below**

Wikipedia : Quadratic Programming

FIN