Lecture 1

12/1/2025

Numerical optimization.

outcome skills (program, based on numerical optimization)

- Building training Neural networks
- Supervised machine learning algorithms
- Reinforcement learning
- choosing Algorithms, measuring its relevance

outcomes (Numerical optimization);

- understanding convexity
- " optimization
- " varients of Gradient Decent Algorithms ?
- momentum based Algorithms ?
- coding & implementation of such algorithm

- Gradient is the slope as a vector (المتجاه)
-

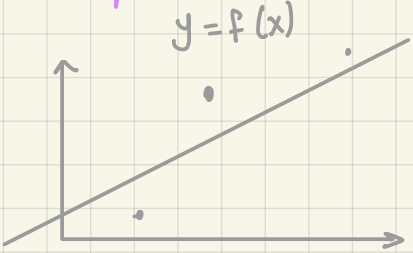
• why numerical optimization?

→ Sometimes the case is too complex to solve it analytically so numerically is better solution (approximation)

⇒ example of analytic solution of "least_squares"

$$\min \left[\sum_i (y_i - \tilde{y}_i)^2 \right]$$

y_i → real, $\tilde{y}_i$ → predicted



for this problem analytic solution exist

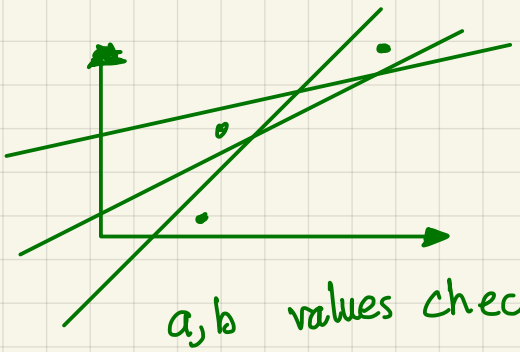
"pseudo_inverse" $\vec{\theta} = x^T (x\, x^T)^{-1} y$

$\vec{\theta}$ contain a, b weights

analytic solution is always the best → get exact value

→ analytic can be hard because of number of inputs or complexity

→ Numeric solution is depending on the approximation



RSS :- sum of squared residuals → البواقي (error)

difference between real-predict to calculate the error

$$RSS = \sum_i (y_i - \tilde{y}_i)^2$$

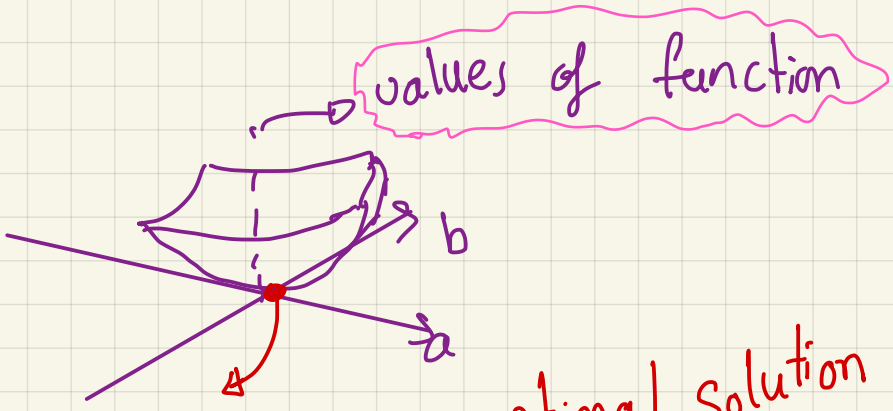
$\tilde{y}_i$ → $f(x) = ax_i + b$

$$= \frac{1}{n} \sum_i (y_i - (a x_i + b))^2$$

→ second norm #

MSE

Supervised Learning (Regression) so I know y, x



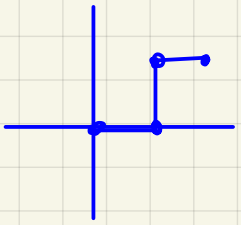
I have one optimal solution

★ Convex ⇒ has global minima

non convex ⇒ has global & Local minima

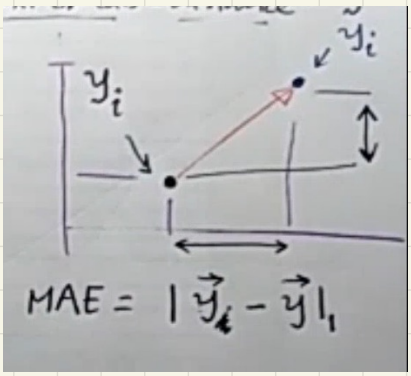
→ Can have more than one global or Local minima

Cost function (Loss function)

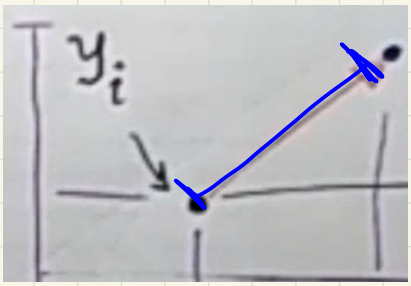


L_1 norm ⇒ MAE → manhatten distance

"taxi or cab distance"



L_2 norm ⇒ MSE ⇒ Eculidian distance بتاعورة



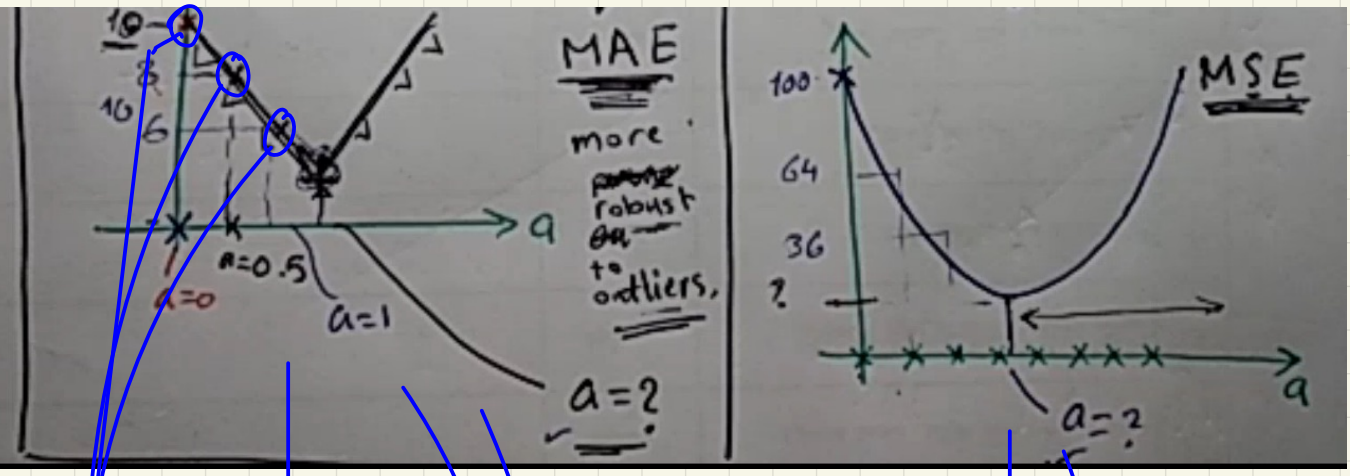
$$\sqrt{(\text{distance 1})^2 + (\text{d2})^2}$$

MAE :- robust outliers

MSE :- error will be much higher "error²"
if there's outliers

$$MAE = \frac{1}{n} \sum_{i=1}^{n} \left| \vec{y}_i - \vec{\tilde{y}}_i \right|$$

ax



محصلة الا بدرور

أقل تأثرا بال outliers

فجوة الا a بتتأثر بال outlier

ميل المنحنى غير معروف عند القاع

بتتأثر أكتر بال outlier

مفضلة أكتر عشان continuous

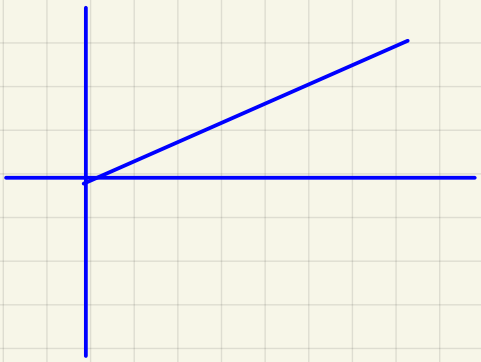
$MSE \propto error^2$

$MAE \propto error$

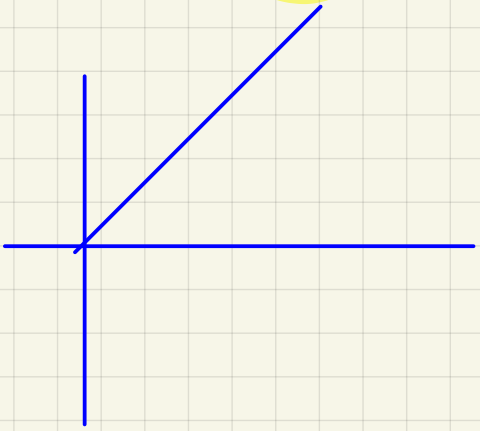
$$RSS = SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$MAE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

التفاعل هو التغير هو الـ slope "الميل"

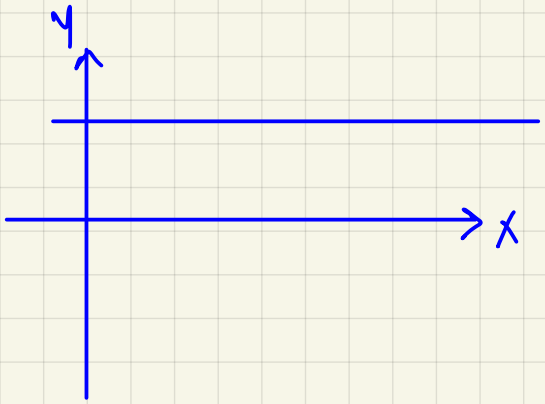


$\frac{\Delta y}{\Delta x}$ = ↓



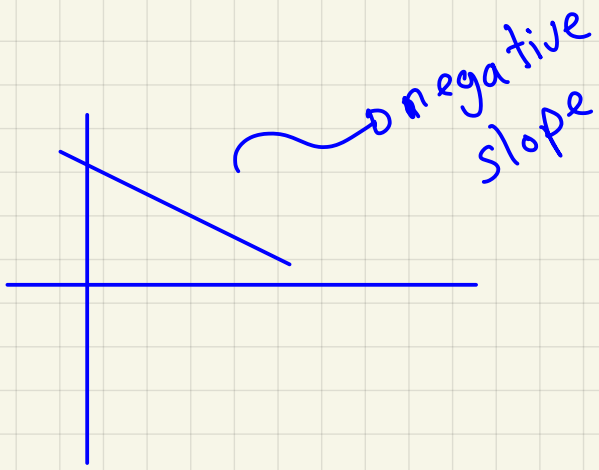
rate of change ↑

$\frac{\Delta y}{\Delta x}$ = ↑



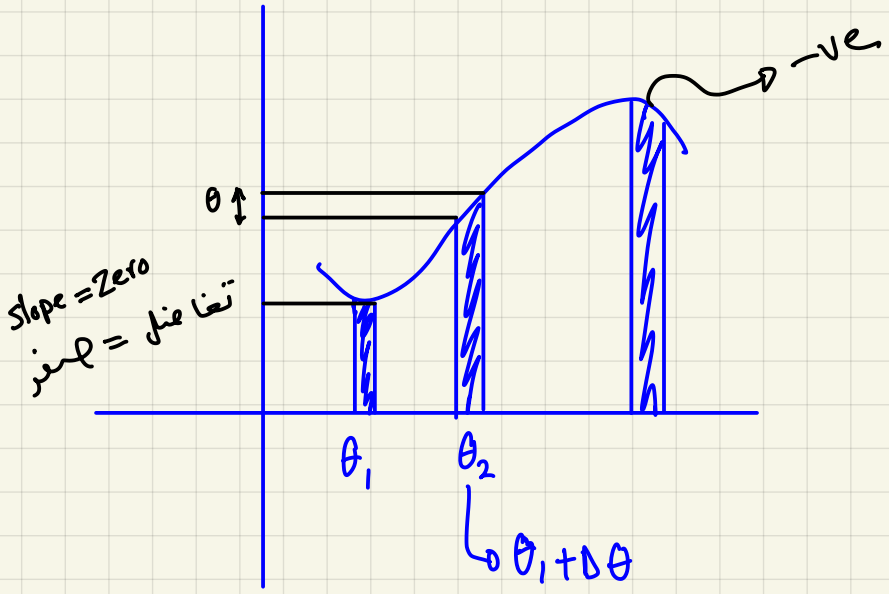
rate of change

$\frac{\Delta y}{\Delta x} = 0$



$\frac{\Delta y}{\Delta x}$ = -ve

rate of change for non linear functions:-



we take small sections to calculate the rate of change

$$\frac{df(\theta)}{d\theta} = \lim_{\Delta\theta \to 0} \frac{f(\theta + \Delta\theta) - f(\theta)}{\Delta\theta}$$

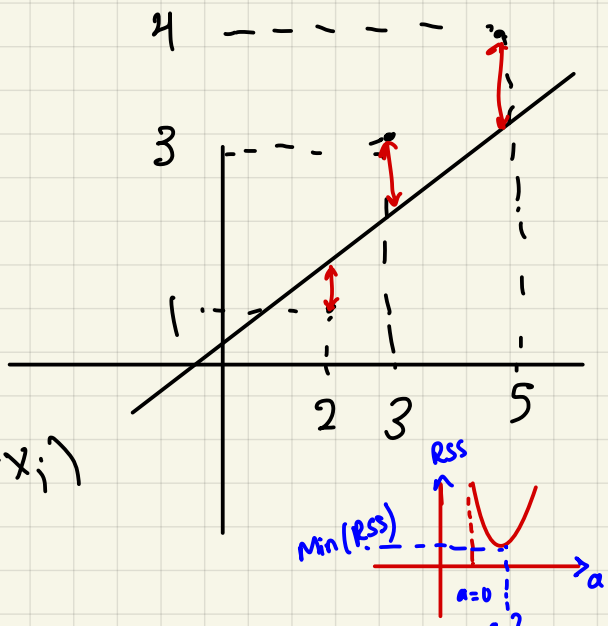
⇒ in V function of MAE there's another problem is that gradient ∇ is always the same

so U is better with the gradient calculation

$$\text{gradient} = \frac{d}{da}\,\text{Cost function}$$

$$\text{cost fun} = \sum_{i}^{5} (y_i - a x_i)^2$$

$$\text{gradient} = 2 \sum_{i=1}^{3} (y - a x_i)(-x_i)$$



$$\text{gradient} = 2\left[(1-2a)(-2) + (3-3a)(-3) + (4-5a)(-5)\right]$$

at $a=0 \Rightarrow \nabla = 2(-2-9-20) = -62$

Update estimate of "a" Let $\alpha = 0.001$

$$a_{new} = a_{old} - \alpha * \text{gradient}$$

$$= 0 - 0.001 \times -62 = \checkmark$$

⋮

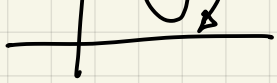
Continue each time update → steps size ($\nabla * \alpha$)
→ gradient
→ new value of a

⇒ you can try different α to get best results
↳ not too large not too small

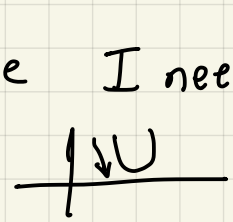
- stop critria can be that gradient is changing by very small value or it stays the same value.

$$a_{new} = a_{old} - \alpha * \nabla$$

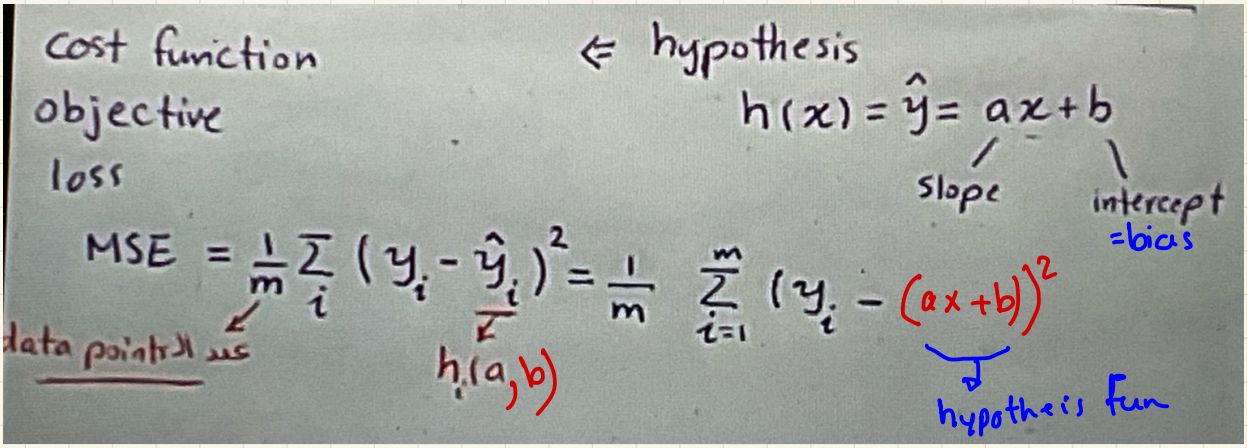
this is to help me if I'm in the positive side I need to decrease the value "move to the negative side"



and if in -ve I need to increase to move to the +ve side



Lecture 2



hypothesis ⟹ فرضية ⟹ $\hat{y}$ function

cost function
= objective function
= Loss function

→ usually refered to as J

$$J(a,b) = \frac{1}{m}\sum_{i=1}^{m} (y_i - (ax+b))^2$$

$$= \frac{1}{m}\sum_{i=1}^{m} (y_i - \bar{y}_i)^2$$

h is the hypothesis fun. ← $h(a,b)$

In MSE we add 2 to $\frac{1}{m}$ to become $\frac{1}{2m}$ to help with the calculations of derivatives to remove the constant

we should continue Repeat update until convergance

$$a_{new} = a_{old} - \alpha \gamma$$
$$b_{new} = b_{old} - \alpha \gamma$$

} update b, a simultaneously

• we need to have a range to α to try to find the best one to our case

---

All that we have discussed was single var.
Let's see Multivariate linear regression

Multivariate linear regression :-

$$h() = \hat{y} = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

intercept / bias

$$h(\theta_0 \cdots \theta_n) = \hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

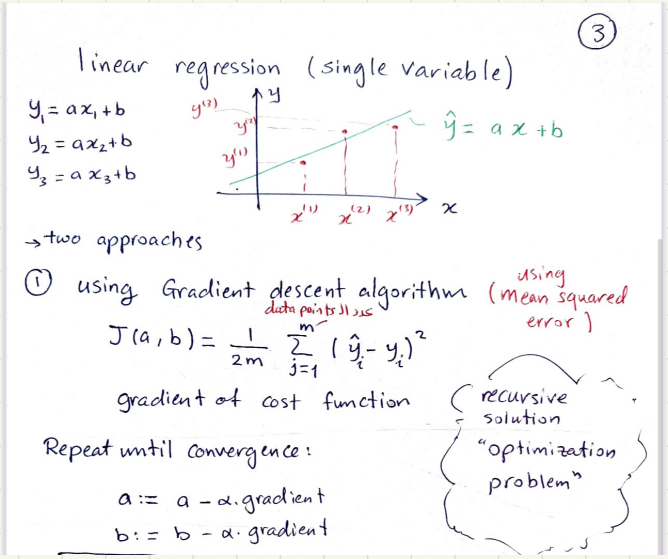
⇒ in Single Variate Linear regression we use the subscript i to reffer to point number but in multivariate we use it to reffer to the i^{th} variable ⟹ $x_i^{(j)}$ ⇒ i^{th} variable, j^{th} data point

we usually write j between () to show that it's not a power

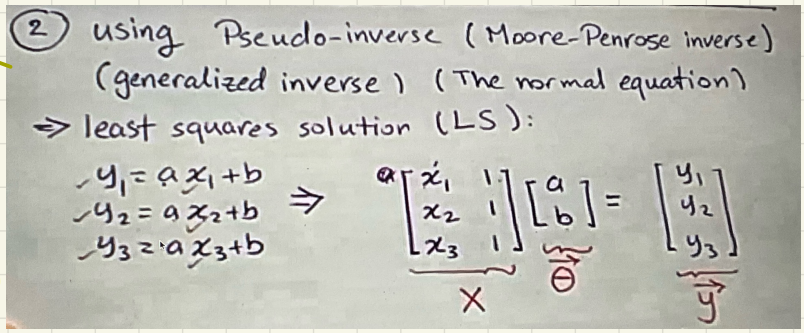
To solve Linear Regression (single variable) :-

we have 2 approches:-

1. gradient descent
2. pseudo inverse



this approach Can not be used if the problem is complex →non convex or we have large number of data points →or X is non invertable



Note if X was square matrix I could just multiply both sides by X^{-1} and solve it directly but because it's not we will use the general form of pseud inverse equation

$$X\vec{\theta} = \vec{y}$$

$$X^T X \vec{\theta} = X^T \vec{y}$$

Square matrix then * by the inv.

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

I

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

Lecture 3

Review :- vanilla Gradient Descent

"Batch GD algorithm"

→ update parameters $\vec{\theta}$

$$\theta_{new} = \theta_{old} - \alpha \cdot \text{gradient of } J(\theta)_{old}$$

$$\theta_{new} = \theta_{old} - \alpha \nabla J(\theta_{old})$$

$$\frac{1}{2m} \sum_{j=1}^{m} (\quad)$$

summation over all data points (all examples in training dataset)

Problems

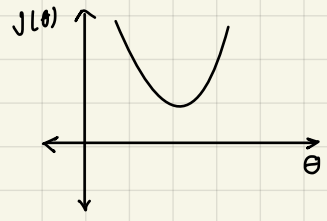
- slow
- large memory needed

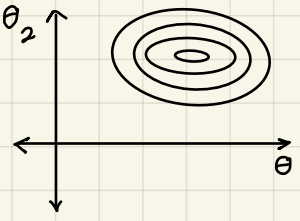
- variations of GD
- problems associated with GD
- acceleration of GD ; → momentum method
  Nestrov method
  Adaptive GD
  Adam "ADAM"

} Next lecture

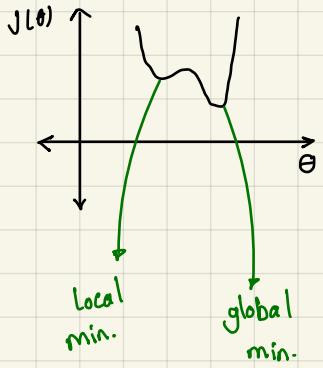
- feature scaling

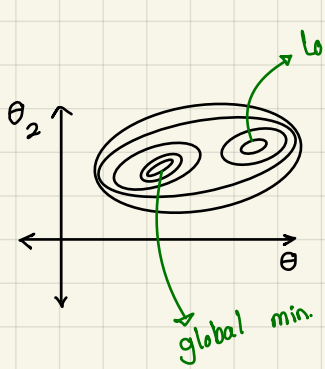
Convex :-





non convex :-





Note :- from the contourplot you can guess the local and global minima by the number of circles or ovals around the minima

---

1 feature scaling

ex $J(\theta_1,\theta_2) = \frac{1}{2m}\sum_{i=1}^{m}\left(y-(\theta_0+\theta_1 x_1+\theta_2 x_2)\right)^2$

zero in this example for simplicity

$\hat{y}$

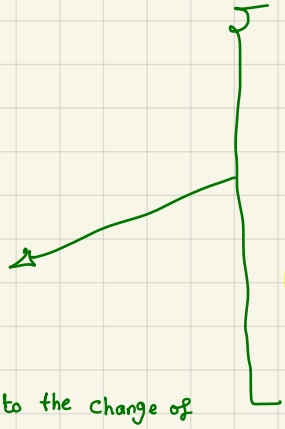
$\theta_0 x_0$, $x_0 = 1$

usually θ_0 is multiplied by x_0 which we always give a value 1

area	# rooms
100	2
⋮	⋮
200	5

$x_1 \in [100, 200]$

$x_2 \in [2, 5]$



Note that range of values of x_1 & x_2 are very different that it will affect the gradient descent calculations

So we need scaling

remember; grad ⇒ derivative ⇒ (J) معدل تغير فـــيـــمـــة

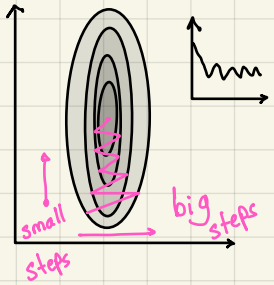
θ parameter إلـــى التغير فى قيمة

the derivative is the rate of change of (J) function to the change of the theta parameters

$\hat{y} = 0 + \theta_1 x_1 + \theta_2 x_2$ (x_1: area)

x_1 has higher scale than x_2

So $\theta_2 \gg \theta_1$



Note

So if the rate of change of one of the x values that will cause a kind of magnifying this θ associated with it and minimize and marginalize the other θ

⇒ it will take alot of steps to reach the optimum not like if we scale the data it will use the optimum number of iterations

- for high values it will get low weight , for Low values it will get high weights . to try to show the effects of each so we need to Scale the input to manage the oscillation

Scaling produce scaled features.

original feature ← $x_1 \in [100, 200]$

Scaled feature ← $|\dot{x}_1| \sim 1$ ⇒ convert values to be arround 1 rather than has very Large variance

0 to 1 or -1 to 1

---

1 Min_Max Normalization :-

$$0 \leqslant x' = \frac{x - x_{min}}{x_{max} - x_{min}} \leqslant 1$$

$$\therefore x' \in [0, 1]$$

x_1	x_1'
100 ← min	0
100	0
150	0.5
200 ← max	1
180	180

$$\frac{x - 100}{200 - 100}$$

it has a big problem that its sensitive to outliers "Low or high"

⇒ Note Normalization can be consider as type of scaling it has types as we discussed

2 mean normalization (standardization)

$$x' = \frac{x - \bar{x}}{\sigma}$$

- Less affected by the outliers
- keep the distances "spread" of data

---

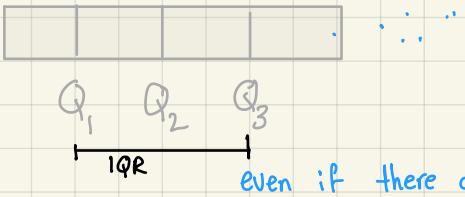
3 Quartiles

$$IQR = Q_3 - Q_1$$

↳ it's a Robust scaling.

$$x' = \frac{x - Q_2}{IQR}$$

IQR ⇒ InterQuartile Range



even if there are outliers the Quartiles are less affected by the outliers

---

there are measures of spread :-

Examples:- IQR, σ, $x_{max} - x_{min}$

↳ standardization always contain divide by one of the measures of spread

مقدار تباعد البيانات وتوزيعها

this is the step of normalization standardization also contains subtract to make it near to 1

Is feature scaling needed in all ML algorithms?

⇒ In data which distance is not important as Naiive bayes classifier or decision tree feature scaling is not importand then

- then it's not in algorithms which are not distance based

Variations of Batch gradient descent

"vanilla

1. ⇒ you multiply $\frac{1}{2m}\sum_{i=1}^{m}$, you do it on all the data which has problems in computation " may solved by matrix operations " but it also needs Large memory

2. The step size is const to all the data which can cause it to stuck somewhere

➡ The possible variations are :-

- stochastic gradient descent
- mini_batch gradient descent

— it is called stochastic because it is randomly initialized , not called like one step or others because of that

— stochastic does not get the advantage of the computation using the vector operations it only fix the problem of the memory as it do not need alot of memoty as the batch GD

— here comes the mini batch GD to balance the both advantages

— pictures in the presentation about the differences of the types of GD

— batch is the lesest one that converge it reach the goal with minimum oscillations but it can stuck in local minima and it has disadvantages that it needs a large memory

— we have some problems : local minima and global miniam which we tried to fix by the variations of the GD and choose the right learning rate

— there are 2 other problems which are :

1. Exploding gradient : the gradient is very large
2. vanishing gradient : the gradient is very slow

— Both problems make training difficult, especially in:

1- Deep neural networks

2- Recurrent Neural Networks (RNNs)

momentum gradient descent:-

The main idea is about the Learning rate (step size) instead of calculate the gradient the normal way:- $\theta_{new} = \theta_{old} - \alpha \nabla J(\theta)$

it will put into consideration the old gradient descent too to still aware of how the gradient is updating should I slow down (smaller step size) or become faster.

The equation of the momentum is:-

$$\theta_{new} = \theta_{old} - (\gamma \cdot \text{old gradient} + \alpha \nabla J(\theta)_{old})$$

- θ_{new}: next step
- γ: to control the amount I will use of the old gradient descent
- $\nabla J(\theta)_{old}$: current gradient

Example:-

- If the gradient of old theta was +ve and the next theta were -ve then when it adds some (γ) of the old ∇ it will take a smaller steps
- If both in the same direction it will add to the step size to make it faster, it means I didn't reach yet I'm still moving the same direction

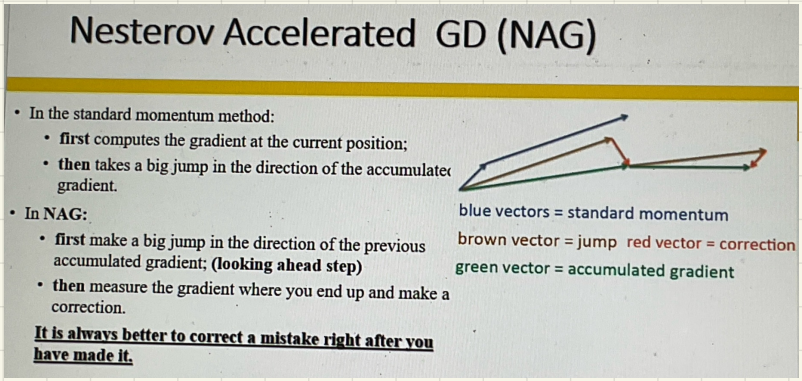
Nesterov:-

.Rather than think of the old gradient it will see the next step by calculating theta temp:-

$$\theta_{temp} = \theta_{old} - \gamma * \text{old gradient}$$

$$\theta_{new} = \theta_{temp} - \alpha \nabla J(\theta)_{temp}$$

$$\text{new gradient} = \gamma \text{ old gradient} + \alpha \nabla J(\theta)_{temp}$$



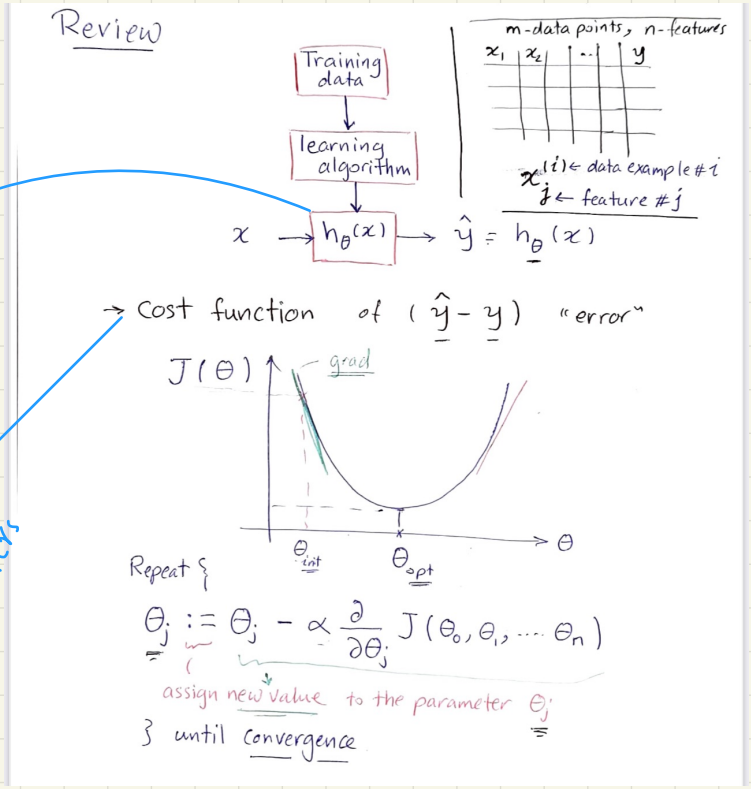
- **Issues with momentum based Gradient descent:**
- Momentum based GD **oscillates** in and out of the minima valley i.e., making **U-turns** near the minima **(however it converges faster vanilla GD).**

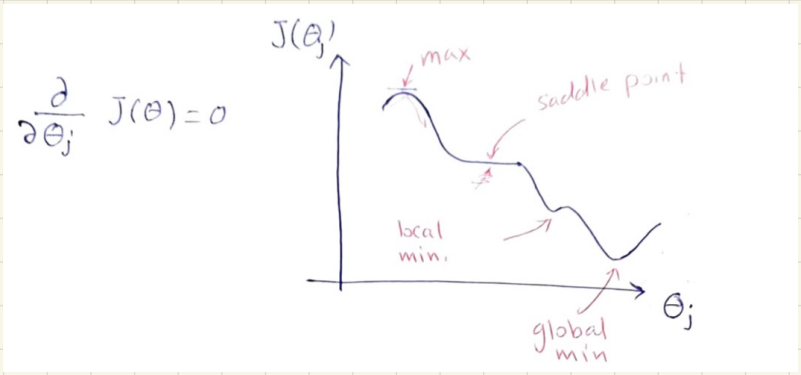
* it corrects the step size before taking the step ⇒ Nesterov

* the optimum is the minibatch because it compensates the advantages of both stochastic & batch gradient descents

Lecture 4

- m ⇒ data points
- n ⇒ features





In these causes the gradient = zero

the max is not a big Problem because in control it called unstable point it will get over it quickly

⇒ we can use any of accelerating gradient descent like momentum, Adam, Adap --etc

⇒ the random initialization helps also because the other initializations can start from a better point

Absorbtion Error :-

* In Some Cases Small numbers can be absorbed by Large numbers . its effect will not be noticed

* Review the floating points

* It's not about find the approximate value , in matrix operations because of the absorbtion effect the full rank matrix may not recognized as full rank because columns has this effect . → that were going to be discussed in the "Numeric Algorithms Course" but it had been removed from ITI programs

that's why we need to add x_0 although it's not actual feature and it always = 1 to be able to belong to $\mathbb{R}^{n+1}$ as thetas

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

$$\vec{\theta} := \vec{\theta} - \alpha \nabla (J(\vec{\theta}))$$

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla (J(\vec{\theta}^{(t)}))$$

$$\vec{\theta}_{K+1}^{*} = \vec{\theta}_K - \alpha \nabla (J(\vec{\theta}_K))$$

different ways to write the equations

→ Vanilla gradient descent

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$$

for all data points m

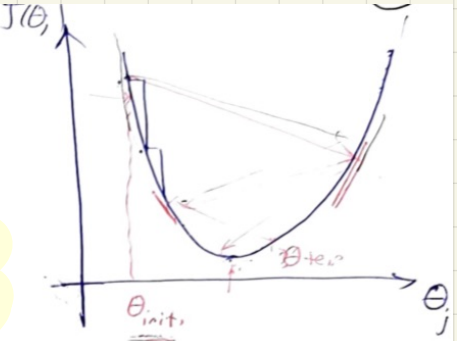
it will not make difference if $\hat{y} - y$ or $y - \hat{y}$ because we usually use $()^2$ or absolute .

- Momemntum GD

$v^{(t)} = \beta v^{(t-1)} + \alpha \nabla J(\theta^{(t)})$

$\theta^{(t+1)} = \theta^{(t)} - v^{(t)}$

Equations



① $\theta^{(0)} = \theta_{init}$

$v^{(0)} = 0 + \alpha \nabla J(\theta^{(0)})$

$\theta^{(2)} = \theta^{(0)} + v^{(1)}$

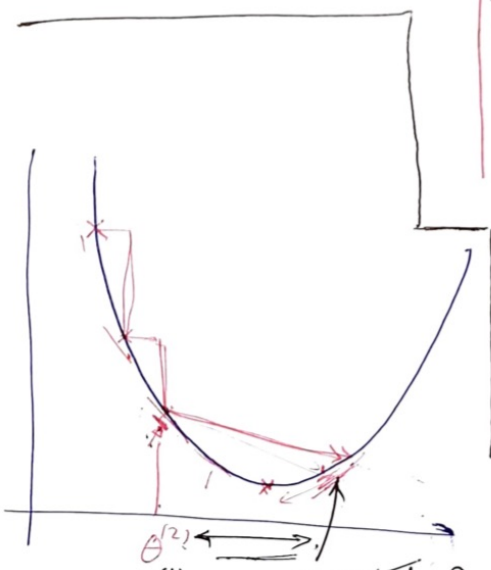
$\theta^{(2)} = \theta^{(1)} - \alpha \nabla J(\theta^{(1)})$

② $v^{(2)} = \beta v^{(1)} + \alpha \nabla J(\theta^{(2)})$

$v^{(2)} = \beta(\alpha \nabla J(\theta^{(1)})) + \alpha \nabla J(\theta^{(2)})$

$\theta^{(3)} = \theta^{(2)} - [\quad]$

exponentially decaying weighted average



Nesterov's ?

expected Predicted (extrapolated) new θ

$\theta^{(t+1)}_{(temp)}$

So that if they are in Some direction speed will be increased if not will be slower

Accelerating GD

- momentum based methods
  - momentum GD
  - Nesterov method

- v(t) is **exponentially** decaying weighted sum, as t increases γV(t−1) becomes smaller and smaller i.e., this equation holds the farther updates by a small magnitude and recent updates by a large magnitude.

↳ exponential because of the power

↳ decaying because γ is less than 1 so the more t "# of data" the less effect it does

Example in slide 36 ⇒ Momentum based GD

$v^t = \gamma \cdot v_{t-1} + \eta \nabla w_t = \gamma^{t-1} \cdot \eta \cdot \nabla w_1 + \gamma^{t-2} \eta \nabla w_2 + \dots + \eta \nabla w_t$

↳ because $v_0 = 0$

⇒ In momentum based GD the gradient can overshoot and skip the minima ⇒ take large step in place it should take a small one Nesterov notice that and his method will try to fix that by calculating the temp to see before I update the parameter what will happen? overshoot, reach the destination .... etc

⇒ Nesterov worked on the momentum based GD but made some changes :-

$$v^{(t)} = \beta v^{(t-1)} + \alpha \nabla J(\theta^{(t)} - \beta v^{(t-1)})$$

$\theta^{(t)} - \beta v^{(t-1)}$ → θ_{temp}

$$\theta^{(t+1)} = \theta^{(t)} - v^{(t)}$$

$$\theta^{(t+1)} = \theta^{(t)} - \beta v^{(t-1)} - \alpha \nabla J(\theta_{temp})$$

$\theta^{(t)} - \beta v^{(t-1)} = \theta_{temp}$

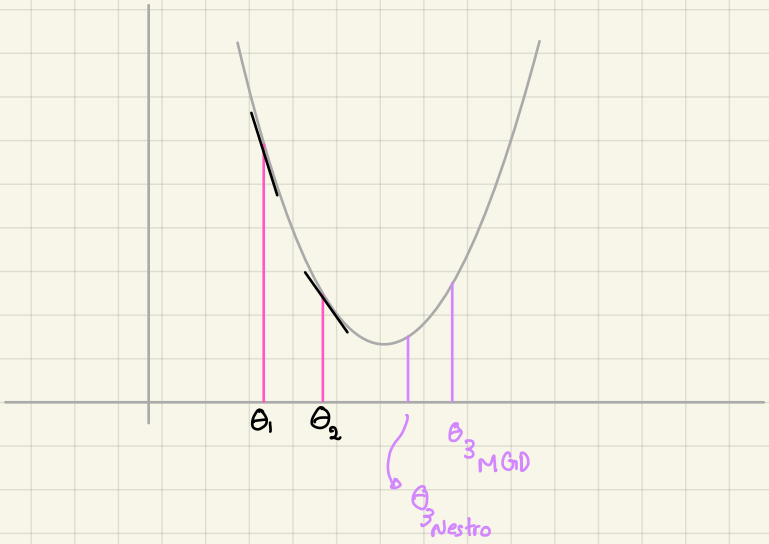
In Nesterov it will reduce the oscillation and prevent the overshoot of Momentum based

The momentum based was :-

$$v^{(t)} = \beta v^{(t-1)} + \alpha \nabla J(\theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - v^{(t)}$$

Comparison between Momentum based GD and Nestrov method :-



momentum

⇒ to update the parameters we take into consideration the old step $\theta^{(t)}$ to calculate the new values $\theta^{(t+1)}$

⇒ $v^{(t)} = \beta\,(\text{grad}\,(J(\theta_1)) + \alpha\,\text{grad}\,(J(\theta_2))$

⇒ (−ve) (−ve)

so $v^{(t)} = -ve \uparrow$

so $\theta_3 = \theta_2 - (-ve \uparrow)$

$\theta_3 >> \theta_2$

Nesterov

⇒ to update the parameters we calculate variable called θ_{temp} which tell us if we continue as we are what will happen? if over shoot it will reduce the amount of the velocity to reduce the overshoot.

⇒ $v^{(t)} = \beta\,(\text{grad}\,(J(\theta_1)) + \text{grad}\,(\theta_{temp})$ (−ve)

$\theta_{temp} = \theta_2 - \beta\,\text{grad}\,(J(\theta_1))\uparrow$ (− (−ve))

$\theta_{temp} >> \theta_2$ ⇒ here comes the advantage of Nestrov

$\theta_3 = \theta_2 - (-ve + ve)$

So it will be reduce the effect

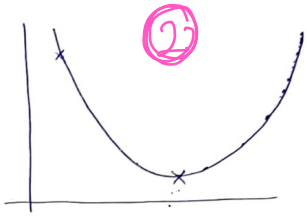
in θ_{temp} we update by θ_{new} not like the v we update by the old value

Continue Accelerating GD but

↳ we will discuss optimizers related to learning rate.

→ Decaying learning rate

Starting with large α, decrease α with time.



I- adaptive gradient (Adagrad)

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \frac{\alpha_j}{\varepsilon + \sqrt{\sum_{k=1}^{t} (\text{grad}(\theta_j^{k-1}))^2}} \text{grad}(\quad)$$

ε: numerical stabilizer (to prevent divide by 0 if gradient = 0)

expressed with velocity term $v^{(t)} = v^{(t-1)} + (\text{grad}(\quad))^2$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\alpha}{\varepsilon + \sqrt{v^{(t)}}} \text{grad}(\quad)$$

- If the gradients are small the α will increase. But it lead to another problem of what if the gradients were large that will reduce the α

adaptive gradient → adapted by adapt the learning rate

→ learning rate

$$\vec{\theta} := \vec{\theta} - \alpha \nabla J(\vec{\theta})$$

↳ learning rate

$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix}$

- Can we use different α for each parameter?

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \vec{\alpha} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha_j \frac{\partial}{\partial \theta_j} J(\vec{\theta}^{(t)})$$

for $\vec{\alpha}$

$$\vec{\theta} := \vec{\theta} - \vec{\alpha} \odot \nabla J(\vec{\theta})$$

vector vector vector $\in \mathbb{R}^{n+1}$

sparse

Hadamard product (element-wise)

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \vec{x} \odot \vec{y} = \begin{bmatrix} x_1 y_1 \\ x_2 y_2 \\ x_3 y_3 \end{bmatrix}$$

2 RMS Prop :- "Root Mean Square Propagation"

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\alpha}{\varepsilon + \sqrt{v^{(t)}}} \text{grad}()$$

$$v^{(t)} = \beta v^{(t-1)} + (1-\beta)(\text{grad}())^2$$

- with time the history of gradients will vanish
- $\beta\uparrow \Rightarrow$ increase the gradient history
- $\beta\downarrow \Rightarrow$ decrease the gradient history

$\Rightarrow \beta$ (beta) is used to control the amount of history

---

3 ADAM :- "Adaptive momentum"

↳ build up momentum with adapting α

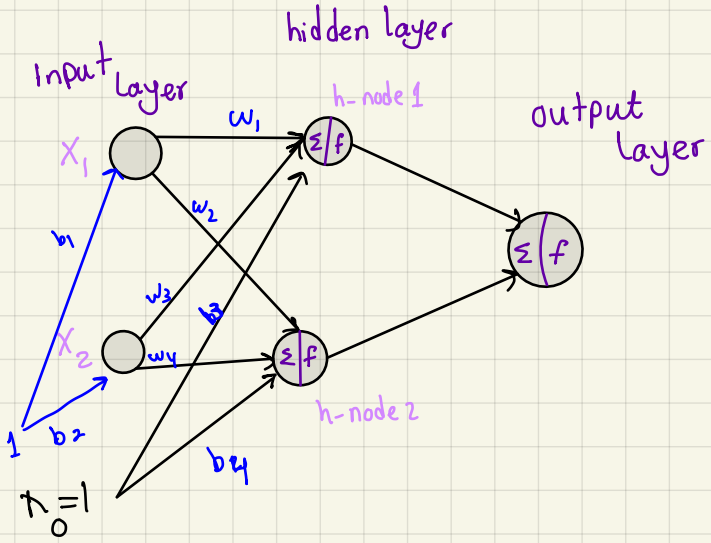
$\Rightarrow$ RMS calculate gradient in the normal way no difference but it uses Adaptive learning with β

$\Rightarrow$ Adam will work on gradient and Adapting Learning rate

will be discussed in next Lecture

Lecture 5

Neural Network:-



Each node contain "perform" 2 things:-

① Σ "summation"

② f ⇒ non linear function because if it's linear the system will be linear Regression and no need for the layers because DL should be for any relation and mostly for non linear Equations, the function called "Activation function"

⇒ all weight are randomly initialized

at h1 :-

$$z_{h_1} = x_1 w_1 + x_2 w_3 + b_3$$

$$out_{h_1} = f(z_{h_1}) = f(x_1 w_1 + x_2 w_3 + b_3)$$

f is → sigmoid ← $$out_{h_1} = \frac{1}{1 + e^{-z_{h_1}}} = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_3 + b_3)}}$$

$$out_{h_2} = f(x_1 w_2 + x_2 w_4 + b_4)$$

$$z_{o_1} = out_{h_1} w_5 + out_{h_2} w_6 + b_5$$

$$\hat{y} = f(z_{o_1}) = \frac{1}{1 - e^{-(z_{o_1})}}$$

we update all the weights

$$w_{5_{new}} = w_{5_{old}} - \alpha \frac{\partial J(\cdot)}{\partial w_5}$$

$$w_{4_{new}} = w_{4_{old}} - \alpha \frac{\partial J(\cdot)}{\partial w_4}$$

⋮

for all parameters (w's and b's)

for example if I use MSE

$$J(w_i) = \frac{1}{2}(\hat{y} - y)^2$$

$$\hat{y} = h_{w,b}(\vec{x})$$

$$\hat{y} = f(z_{o_1}) = \frac{1}{1 + e^{-z_{o_1}}}, \quad \frac{1}{1 + e^{-(Out_{h_1} w_5 + Out_{h_2} w_6 + b_5)}}$$

$$\frac{\partial J(\cdot)}{\partial w_1} = \frac{\partial J(\cdot)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_{o_1}} \times \frac{\partial z_{o_1}}{\partial w_1} \Rightarrow \text{Chain Rule}$$

$$\frac{\partial J(\cdot)}{\partial w_2} = \frac{\partial J(\cdot)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_{o_1}} * \frac{\partial z_{o_1}}{\partial w_2}$$

⋮

and so on

For example $\frac{\partial J(\cdot)}{\partial w_1} = 0.1$

so update $w_{1_{new}} = w_{1_{old}} - \alpha * 0.1 = w_{1_{new}}$

Exponentially weighted moving average (EWMA):-

⇒ exponentially decaying weighted sum is a special case of Exponentially weighted moving average

⇒ exponentially decaying weighted sum decay with time but moving average not always decreasing

⇒ moving average is about having a window size and calculate the avg.

⇒ weighted moving average ⇒ give new values higher weights and old points small weights

⇒ exponentially ⇒ multiply the weights as exponential to give different weights to data ⇒ (t ⇒ high) & (t-1 ⇒ less) & (t-1 ⇒ less than t-1) ....etc

$$v^{(t)} = \beta\, v^{(t-1)} + (1-\beta)x$$

$$v^{(0)} = 0$$

$$v^{(1)} = \beta\, v^{(0)} + (1-\beta)x^{(1)}$$

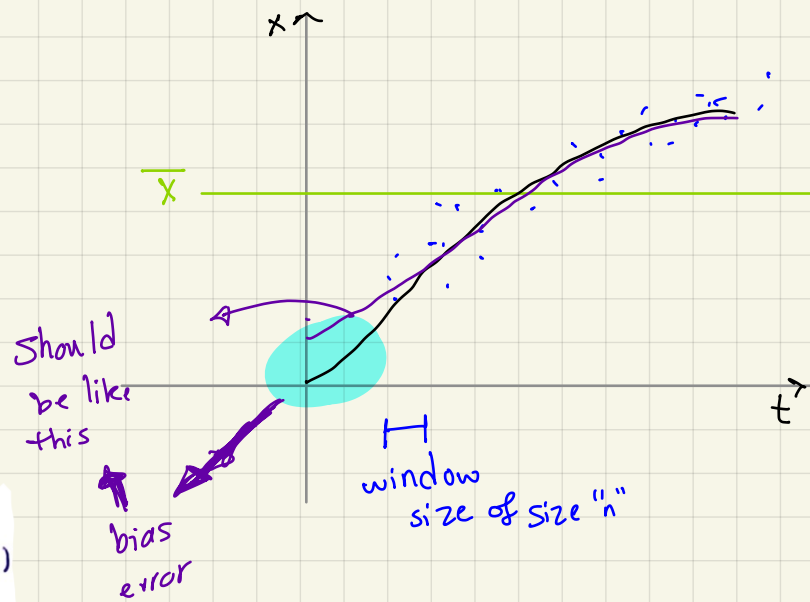
$$v^{(2)} = \beta\, v^{(1)} + (1-\beta)\, x^{(2)}$$

$$= \beta(1-\beta)x^{(1)} + (1-\beta)\, x^{(2)}$$

$$v^{(3)} = \beta v^{(2)} + (1-\beta)\, x^{(3)}$$

$$v^{(3)} = \beta^2(1-\beta)x^{(1)} + \beta(1-\beta)x^{(2)} + (1-\beta)x^{(3)}$$

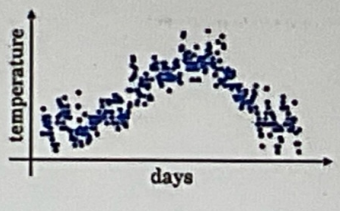
$$v^{(4)} = \beta^3(1-\beta)x^{(1)} + \beta^2(1-\beta)x^{(2)} + \beta(1-\beta)x^{(3)} + (1-\beta)x^{(4)}$$



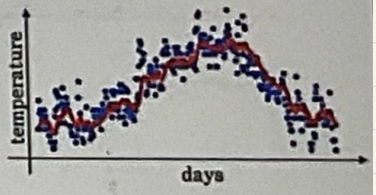
↳ β is less than 1 so β^3 is very low

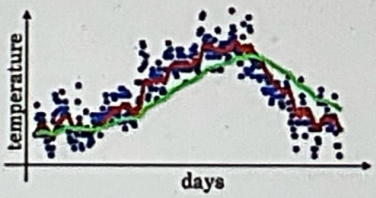
Exponentially Weighted Moving Average (EWMA)

- $v_t = \beta * v_{t-1} + (1 - \beta) T_t$
- Averaging over $\frac{1}{1-\beta}$ days
- β=0.9 averaging over 10 days
- β=0.98 averaging over 50 days
- β=0.5 averaging over 2 days

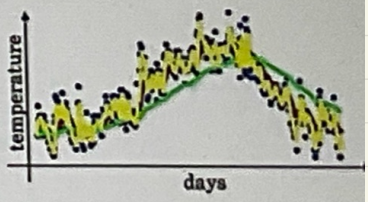


$\beta = 0.9$





$\beta = 0.98$



the green line is below the data because the early points were small and increased later because data in the middle are high

→ because $v_0 = zero$ a problem appears which is bias needs bias correction it happens at the begining of the curve "large error"

→ For bias correction $\hat{v}(t) = \frac{v(t)}{1-\beta^t}$ ← number of steps

↳ for $\hat{v}(1) = \frac{1}{1-\beta^1}$

$\hat{v}(2) = \frac{1}{1-\beta^2}$

⋮

by time t will be very large

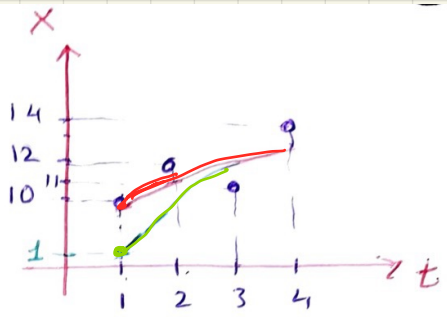
and $\hat{v}$ will be very small

"eleminated

Bias Correction

Bias correction

let $\beta = 0.9$



without Bias correction

$v^{(0)} = 0$ ←

$v^{(1)} = \beta v^{(0)} + (1-\beta) x^{(1)}$

$= 0.9 \times 0 + 0.1 \times 10$

$= 1$

$v^{(2)} = \beta v^{(1)} + 0.1 \times 12$

$= 1 + 1.2$

with Bias corrected ←

$\hat{v}^{(1)} = \frac{v^{(1)}}{1-\beta^1}$

$= \frac{1}{1-0.9^1} = \frac{1}{0.1} = 10$

$\hat{v}^{(5)} = \frac{v^{(5)}}{1-\beta^5}$

$\approx v^{5}$

for large time index.

$1-\beta^t \approx 1$

$\hat{v}^{(t)} \approx v^{(t)}$

Momentum Review:-

$$v^{(t)} = \beta v^{(t-1)} + \alpha \nabla (J(\theta^{(t)})) \longrightarrow \text{I}$$

$$\theta^{(t+1)} = \theta^{(t)} - v^{(t)} \longrightarrow \text{II}$$

Quick Note :-
you might find

Some references write $v^{(t+1)} = \beta v^{(t)} + \alpha \nabla (J(\theta^{(t)}))$

and others $v^{(t)} = \beta v^{(t-1)} + \alpha \nabla (J(\theta^{(t)}))$

No problem

from I & II

$$\theta^{(t+1)} = \theta^{(t)} - \beta v^{(t-1)} - \alpha \cdot \text{grad}$$

EWMA Form:-

may be defined as:-

$$v^{(t)} = \beta v^{(t+1)} + (1-\beta) \nabla (J(\theta^{(t)}))$$

- velocity, $v^{(t)}$
- momentum, $m^{(t)}$

$$\theta^{(t+1)} = \theta^{(t)} - \alpha v^{(t)}$$

$$\text{or } \theta^{(t+1)} = \theta^{(t)} - \alpha \beta v^{(t+1)} - \alpha (1-\beta) \cdot \text{grad})$$

$\Rightarrow$ Because α is multiplied by a small factor in these examples you will need to use large α

1. Adagrad

- 📌 **Adapts** learning rate per parameter
- ❗ Learning rate keeps **shrinking too much** over time
- Good for **sparse data** (e.g., NLP)

Update rule:

$$\theta := \theta - \frac{\alpha}{\sqrt{G + \epsilon}} \cdot \nabla_\theta$$

- G: sum of squared gradients

---

2. RMSProp

- Fixes Adagrad's shrinking problem
- Uses **moving average** of squared gradients
- Good for **non-stationary** problems (e.g., RL)

Update rule:

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla_\theta^2$$
$$\theta := \theta - \frac{\alpha}{\sqrt{v_t + \epsilon}} \cdot \nabla_\theta$$

---

3. Adam

- Combines **Momentum + RMSProp**
- Tracks both **mean** and **variance** of gradients
- Works well in **most deep learning problems**

Update rule:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla_\theta^2$$
$$\theta := \theta - \alpha \cdot \frac{m_t}{\sqrt{v_t} + \epsilon}$$

✅ Summary Table:

Optimizer	Learns rate adapts?	Uses momentum?	Good for
Adagrad	Yes (but shrinks)	No	Sparse data
RMSProp	Yes (with avg)	No	Non-stationary tasks
Adam	Yes	Yes	Most deep learning tasks

✅ EWMA: Exponentially Weighted Moving Average

- A method to **smooth values over time**
- New values are weighted **less** as they get older

Formula:

$$v_t = \beta v_{t-1} + (1 - \beta) x_t$$

- x_t: current value
- v_t: smoothed average
- β: decay factor (e.g., 0.9 or 0.99)

✅ Momentum in Gradient Descent

- Uses **EWMA on gradients**
- Helps the optimizer move **faster in right direction**, and **smooths noise**

Formula (same style):

$$m_t = \beta m_{t-1} + (1 - \beta) \nabla_\theta$$

🎯 So Yes:

Momentum = EWMA of gradients

They are **conceptually the same**, just applied in **different contexts**.

Let me know if you want a visual or code example!

Review RMSprop.

$$v^{(t)} = \beta\, v^{(t-1)} + (1-\beta)(\text{grad}_i)^2$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\alpha}{\sqrt{v^{(t)}} + \epsilon} \nabla(J(\theta^{(t)}))$$

Adam (Adaptive Momentum)

EWMA momentum

$$m^{(t)} \equiv v_1^{(t)} = \beta_1 v_1^{(t-1)} + (1-\beta_1) \underbrace{\nabla(J(\theta^{(t)}))}_{\text{grad.}}$$

$$v_2^{(t)} = \beta_2 v_2^{(t-1)} + (1-\beta_2) \left(\nabla J(\theta^{(t)})\right)^2$$

RMS prop.

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\alpha}{\sqrt{\hat{v}_2^{(t)}} + \epsilon} \hat{v}_1^{(t)}$$

bias-corrected $v_2^{(t)}$

bias-corrected $v_1^{(t)}$

commonly $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\epsilon = 1 \times 10^{-8}$

Newton's method:-

- ⇒ alternative for GD
- ⇒ Sometimes it's better sometimes no
- ⇒ it needs convex function if non it may stuck in some local minima

⇒ it's more about find the zero of the function

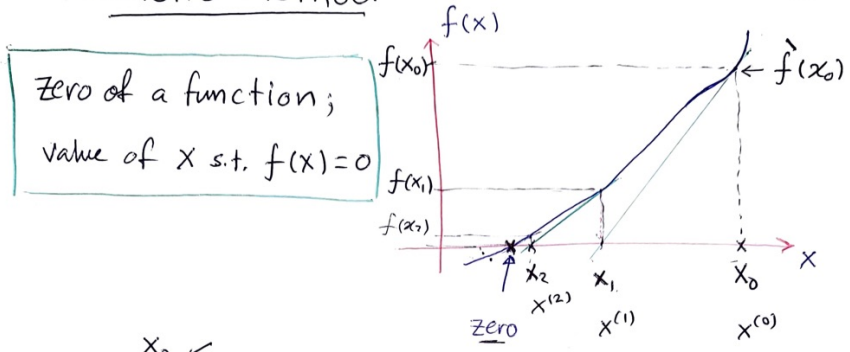
value of x that $f(x) = 0$

Newton's Method

(7)

Zero of a function;

value of X s.t. $f(x) = 0$



x_0

$$\Rightarrow \hat{f}(x_0) = \frac{\Delta Y}{\Delta X} = \frac{f(x_0) - 0}{x_0 - x_1}$$

$$\hat{f}(x_0) = \frac{f(x_0)}{x_0 - x_1}$$

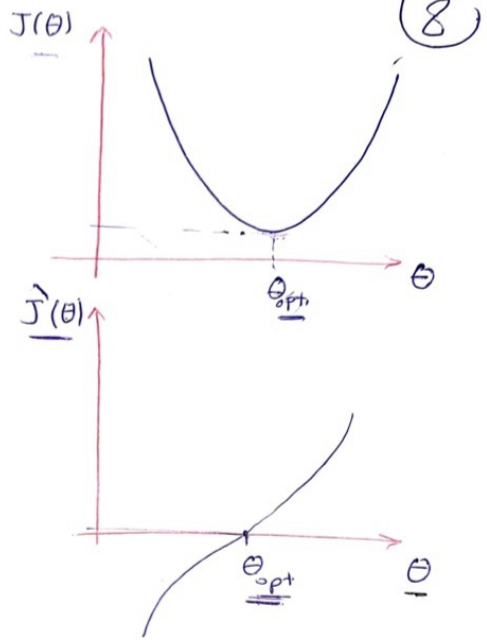
$$\Rightarrow x_0 - x_1 = \frac{f(x_0)}{\hat{f}(x_0)} \Rightarrow x_1 = x_0 - \frac{f(x_0)}{\hat{f}(x_0)}$$

$$x_2 = x_1 - \frac{f(x_1)}{\hat{f}(x_1)}$$

$$x^{(t)} = x^{(t-1)} - \frac{f(x^{(t-1)})}{\hat{f}(x^{(t-1)})}$$

the derivative = zero in 3 cases

- max point
- min //
- saddle //



Zero of $J'(\theta)$

$\Rightarrow$ minimum of $J(\theta)$

(or maximum, or saddle point, or local minimum!)

that's the problem of Newton's method

→ use Newton's method to find zero of $J'(\theta)$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{J'(\theta^{(t)})}{J''(\theta^{(t)})}$$

⇒ In Newton if there are more than θ we will use gradient to perform partial derivative

for multivariate cases

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \frac{\nabla J(\theta^{(t)})}{?}$$

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - (H^{-1}) \nabla J(\theta^{t})$$

GD $O(n)$ complexity ? $O(n^3)$

Newton's method advantages:-

- doesn't need learning rate → use only the slope
- converge quickly as number of steps

But it's complexity is much higher than GD → O(n)

Why the newton uses first and second derivatives what is the usage of each

Great question! Here's the simple reason:

✅ Why Newton uses both:

1. First derivative (Gradient):

- Tells the **direction** to move (like gradient descent)
- Shows **slope** of the function

2. Second derivative (Hessian):

- Tells how the slope is **changing** (curvature)
- Helps adjust **step size**
  - If curve is steep → take small step
  - If curve is flat → take larger step

✅ Purpose of using both:

- Gradient shows where to go
- Hessian shows **how fast** to go in that direction
- Together they make updates **faster and more accurate**

✅ Gradient Descent:

- Uses **first derivative** (gradient)
- Moves in the direction of **steepest descent**
- Goal: **minimize** the loss function
- Slower, but easier to compute

✅ Newton's Method:

- Uses **first + second derivatives** (gradient + Hessian)
- Adjusts steps using **curvature**
- Also used to **find minimum**, not just zero
- Faster convergence, but more complex

🎯 Key Differences:

Feature	Gradient Descent	Newton's Method
Uses	First derivative	First + second derivatives
Goal	Find minimum	Find minimum (not just zeros)
Speed	Slower	Faster (when Hessian is accurate)
Cost	Low	High (need to invert Hessian)

Remember

gradient $\nabla f = \begin{bmatrix} \partial f/\partial \theta_1 \\ \vdots \\ \partial f/\partial \theta_n \end{bmatrix}$

e.g., $f(x,y) \Rightarrow \nabla f = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \end{bmatrix}$

Hessian $H \equiv \begin{bmatrix} \frac{\partial f}{\partial x^2} & \frac{\partial f}{\partial x \partial y} \\ \frac{\partial f}{\partial x \partial y} & \frac{\partial f}{\partial y^2} \end{bmatrix}$

$H = \begin{bmatrix} \vdots \\ H_{i,j} \\ \vdots \end{bmatrix}$

$H_{i,j} = \frac{\partial f}{\partial \theta_i \partial \theta_j}$

$\nabla J(\theta_1 \to \theta_n) = \begin{bmatrix} \partial \\ \\ \end{bmatrix}$

- Quasi-Newton' methods
- Secant method
- BFGS (Broyden, Fletcher, Goldfarb, Shanno,)

→ for further search

note

note that the same method can be expressed in different forms.

e.g., momentum method (standard form)

$$v^{(t)} = \beta v^{(t-1)} + \alpha \nabla J(\theta^{(t)})$$
$$\theta^{(t+1)} = \theta^{(t)} - v^{(t)}$$

OR

$$v^{(t+1)} = \beta v^{(t)} + \alpha \nabla J(\theta^{(t)})$$
$$\theta^{(t+1)} = \theta^{(t)} - v^{(t+1)}$$

OR

$$v^{(t+1)} = \beta v^{(t)} - \alpha \nabla J(\theta^{(t)})$$
$$\theta^{(t+1)} = \theta^{(t)} + v^{(t+1)}$$

and so on, ...

Regardless of how it is written, the parampeters are updated in the same manner.