

Lecture 1

12/11/2025

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 ↗
- " variants of Gradient Descent Algorithms
- momentum based Algorithms ?
- Coding & implementation of such algorithm

- Gradient is ^{the} slope as a vector (jölküll)

-

• why numerical optimization?

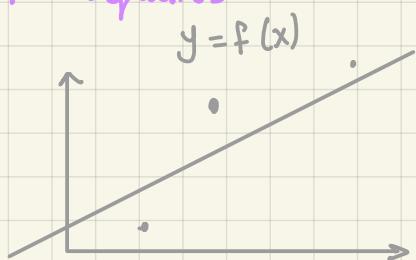
→ sometimes the case is too complex to solve it analytically

so numerically is better solution (approximation)

⇒ example of analytic solution of "least-squares"

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

real + predicted



for this problem analytic solution exist

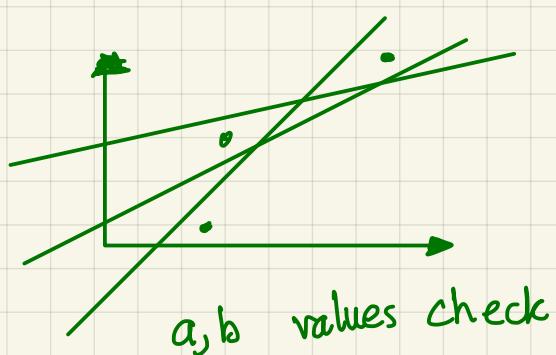
"pseudo-inverse" $\vec{\theta} = x^T (x x^T)^{-1} y$

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

difference between real - predict to calculate the error

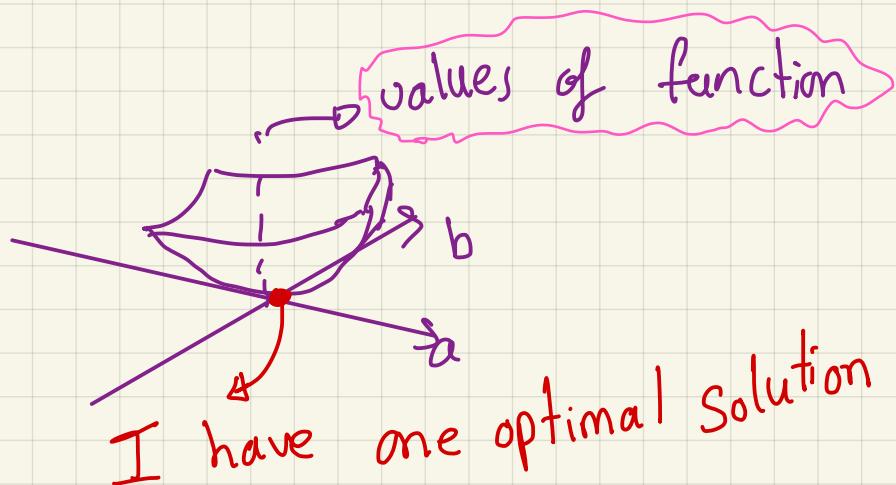
$$\text{RSS} = \sum_i (y_i - \tilde{y}_i)^2$$

\tilde{y}_i $f(x) = ax_i + b$

$$= \frac{1}{n} \sum_i (y_i - (ax_i + b))^2 \rightarrow \text{second norm } \| \cdot \|$$

MSE

Supervised Learning (Regression) so I know y, x

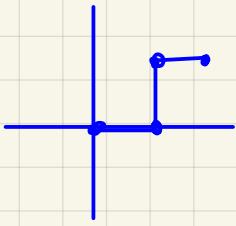


* Convex \Rightarrow has global minima

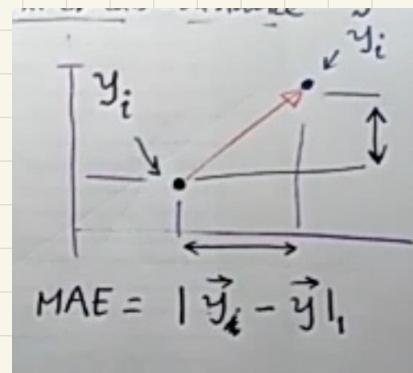
non convex \Rightarrow has global & Local minima

Can have more than one global or local minima

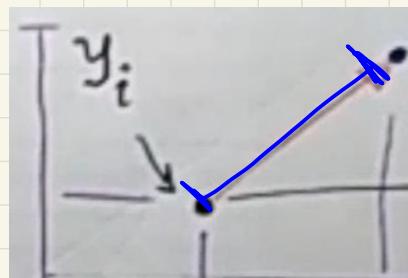
Cost function (Loss function)



L_1 norm \Rightarrow MAE \rightarrow manhattan distance
"taxi or cab distance"



L_2 norm \Rightarrow MSE \Rightarrow Euclidian distance مسافة رأسية

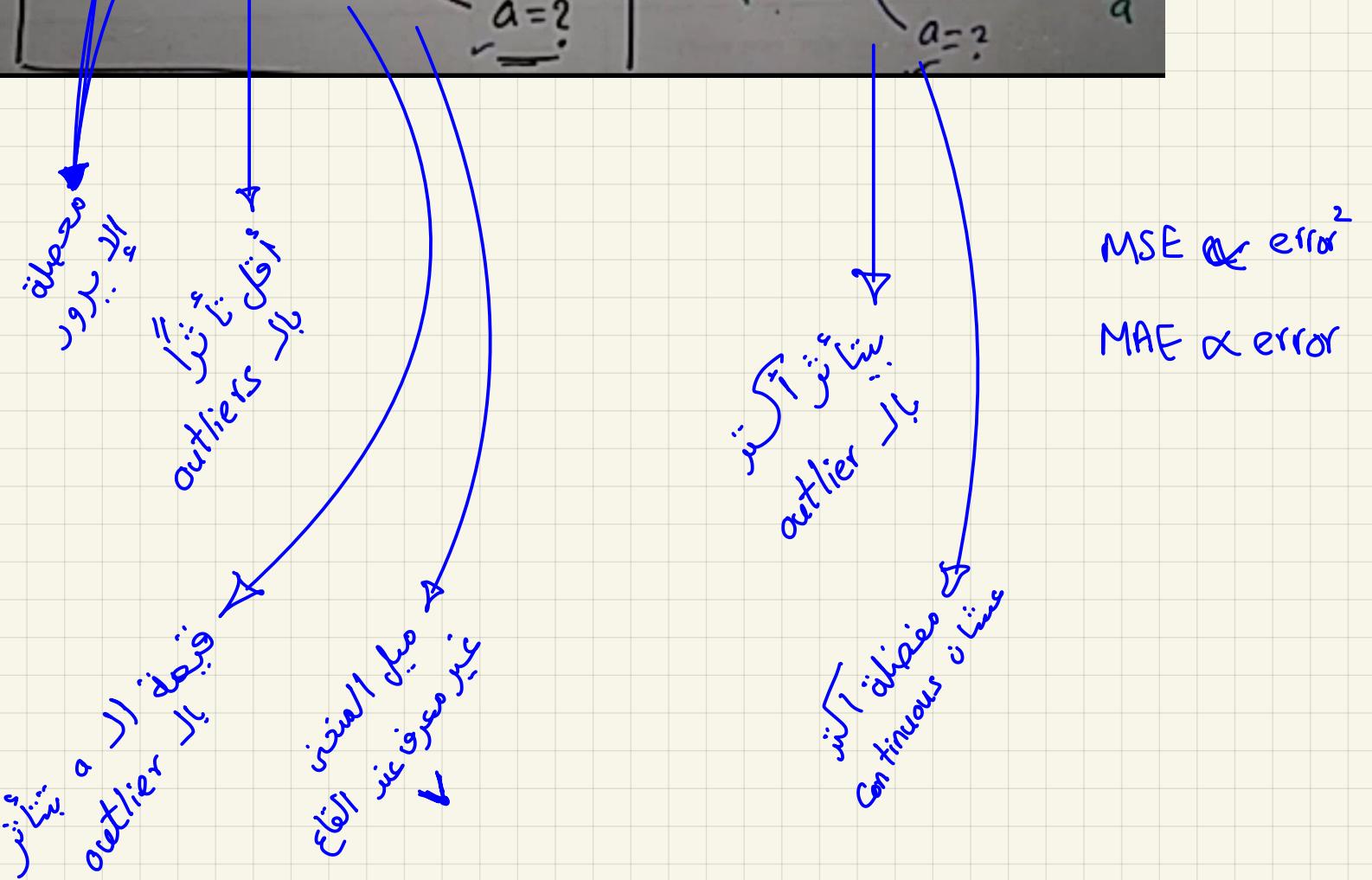
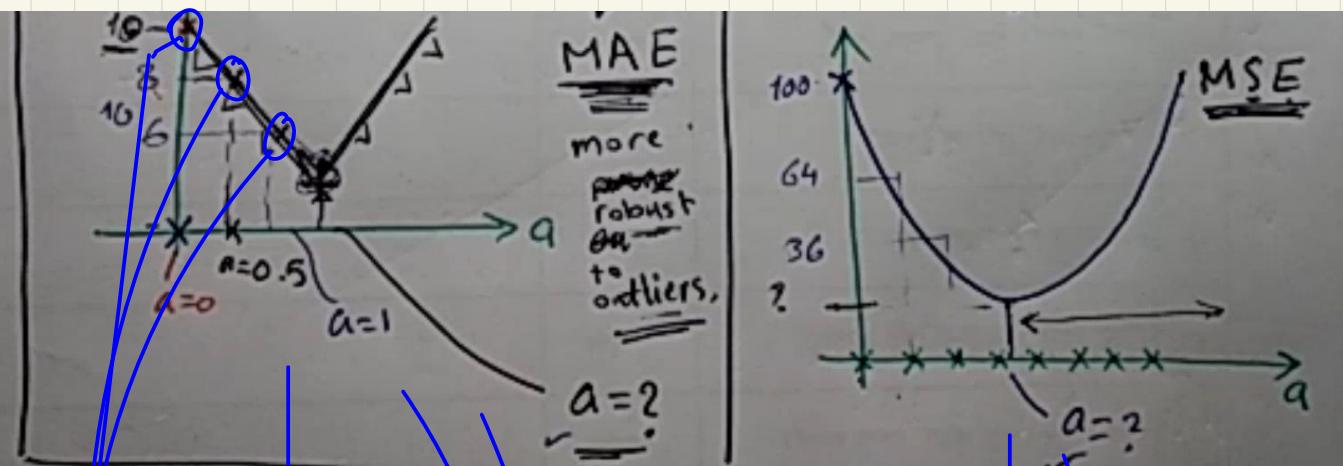


$$\sqrt{(\text{distance}_1)^2 + (\text{distance}_2)^2}$$

MAE :- robust outliers

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

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



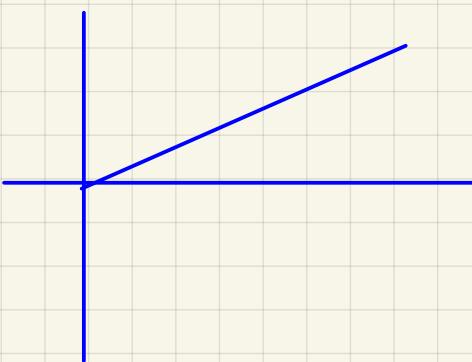
$$RSS = SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

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

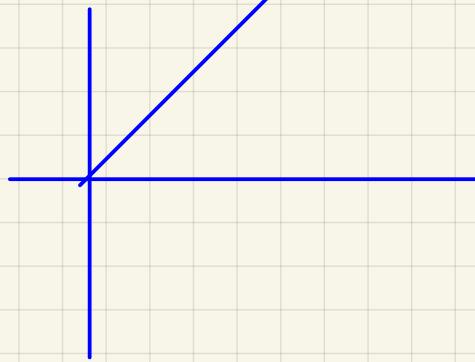
"الميل"

Slope

ـ الميل هو التغير

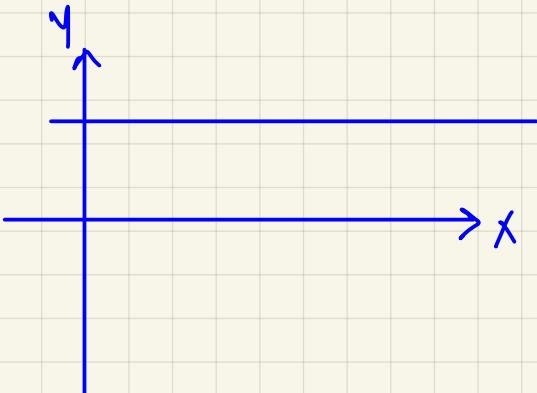


$$\frac{\Delta y}{\Delta x} = \downarrow$$



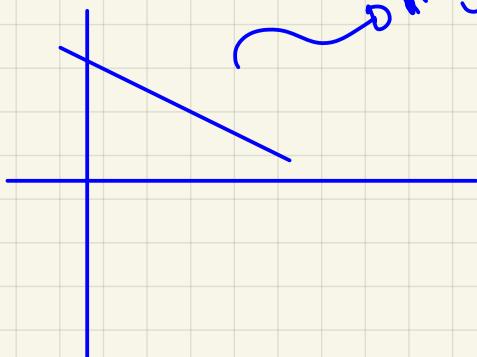
rate of change ↑

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



rate of change

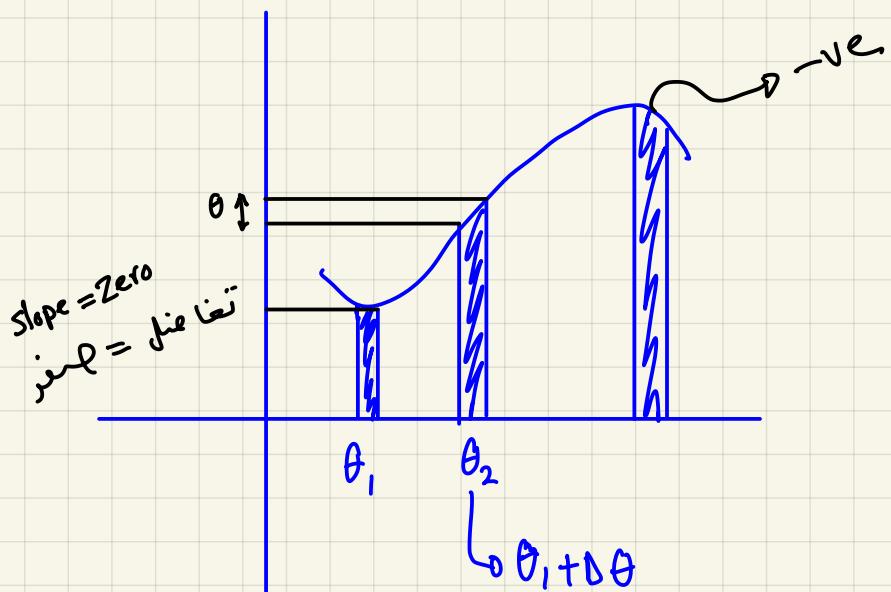
$$\frac{\Delta y}{\Delta x} = 0$$



negative slope

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

rate of change for non linear functions:-



we take small sections to calculate the rate of change

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

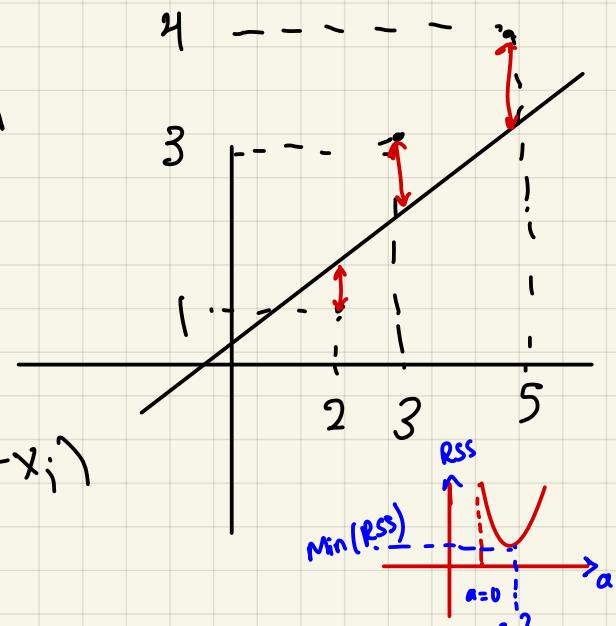
→ in V function of MAE there's another problem
is that gradient γ is always the same

so U is better with the gradient calculation

$$\text{gradient} = \frac{d}{da} \text{ cost function}$$

$$\text{cost fun} = \sum_i (y_i - ax_i)^2$$

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



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

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

Update estimate of "a"

let $\alpha = 0.001$

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

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

⋮

continue each time update

\rightarrow steps size ($\nabla * \alpha$)
 \rightarrow gradient
 \rightarrow new value of a

\Rightarrow 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.

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



↳ 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

cost function objective loss <u>data points</u> we use	\Leftrightarrow	hypothesis $h(x) = \hat{y} = ax + b$ <div style="display: flex; justify-content: space-around; align-items: center;"> a slope b intercept = bias </div> $\text{MSE} = \frac{1}{m} \sum_i (y_i - \hat{y}_i)^2 = \frac{1}{m} \sum_{i=1}^m (y_i - (ax + b))^2$ <p style="text-align: right; margin-top: -20px;">$\underbrace{\quad}_{\text{hypothesis fun}}$</p>
---	-------------------	---

hypothesis \Rightarrow \hat{y} \Rightarrow \hat{y} function

cost function = objective function = Loss function	$\left\{ \begin{array}{l} \\ \\ \end{array} \right.$	usually referred to as J $J(a, b) = \frac{1}{m} \sum_{i=1}^m (y_i - (ax + b))^2$ $= \frac{1}{m} \sum_{i=1}^m (y_i - \bar{y}_i)^2$ <p style="text-align: center; margin-top: -20px;"> $\underbrace{h \text{ is the hypothesis fun.}}_{h(a, b)}$ </p>
--	--	---

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 convergence

$$\begin{aligned}
 a_{\text{new}} &= a_{\text{old}} - \alpha \nabla \\
 b_{\text{new}} &= b_{\text{old}} - \alpha \nabla
 \end{aligned}
 \quad \left\{ \begin{array}{l} \text{update } b, a \text{ simultaneously} \\ \text{update } b, a \text{ simultaneously} \end{array} \right.$$

- we need to have arrange 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:-

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

intercept/bias

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

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

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

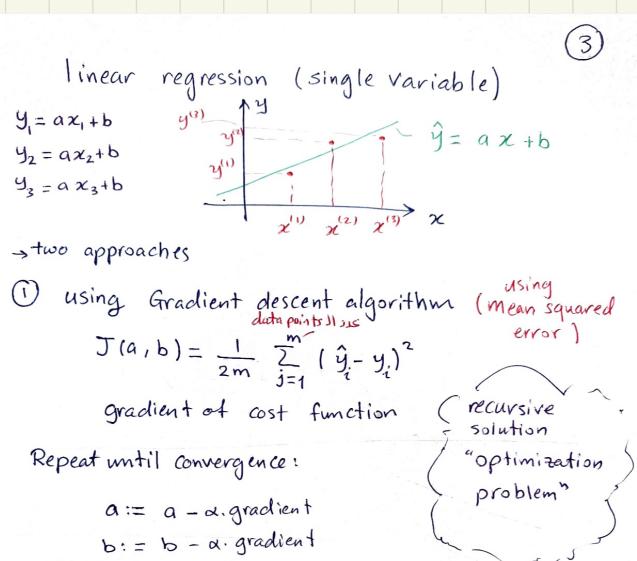
To Solve Linear Regression (Single Variable) :-

we have 2 approaches:-

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 invertible



② using Pseudo-inverse (Moore-Penrose inverse)
(generalized inverse) (The normal equation)

⇒ least squares solution (LS):

$$\begin{aligned} y_1 &= ax_1 + b \\ y_2 &= ax_2 + b \\ y_3 &= ax_3 + b \end{aligned} \Rightarrow \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix}}_X \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\vec{\theta}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}}_{\vec{y}}$$

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

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

Square matrix then * by the inv.

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

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

I

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

Lecture 3

Review :- vanilla Gradient Descent

"Batch GD algorithm"

→ update parameters $\vec{\theta}$

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

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

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

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

Nesterov method

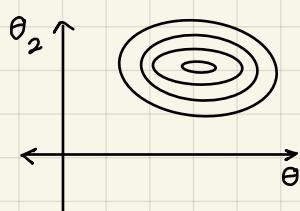
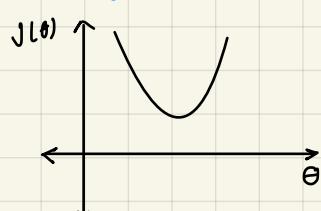
Adaptive GD

Adam "ADAM"

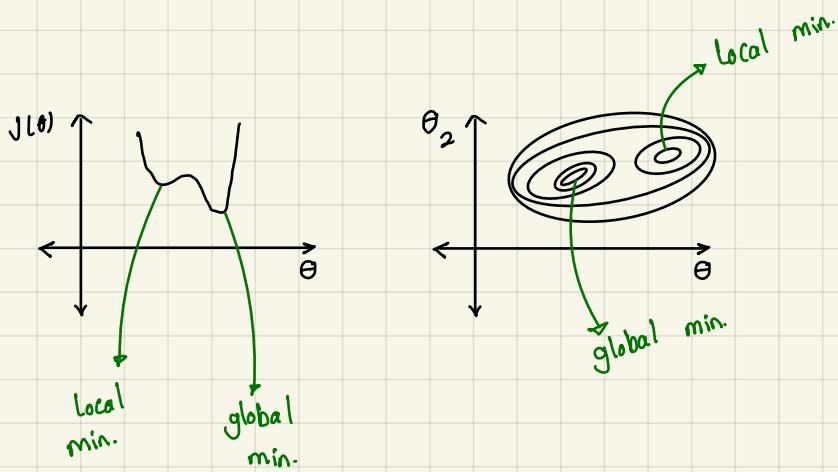
} Next lecture

- 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

II feature scaling

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

Zero in this example for simplicity

y

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

$\theta_0, x_0, \theta_1, x_1, \theta_2, x_2$

area	# rooms
100	2
:	:
200	5

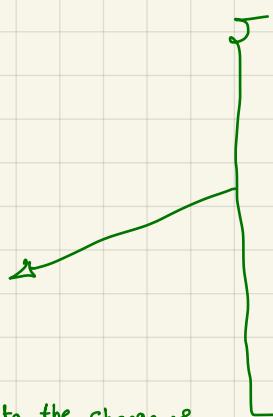
$$x_1 \in [100, 200]$$

$$x_2 \in [2, 5]$$

remember; grad \Rightarrow derivative \Rightarrow (J) مدل تغییر فتحیه

θ parameter تغییر فتحیه

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



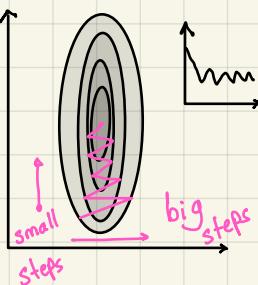
Note that range of values of x_1 & x_2 are very different that it will affect the gradient descent calculations

So we need scaling

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

x_1 has higher scale than x_2

so $\theta_2 \gg \theta_1$



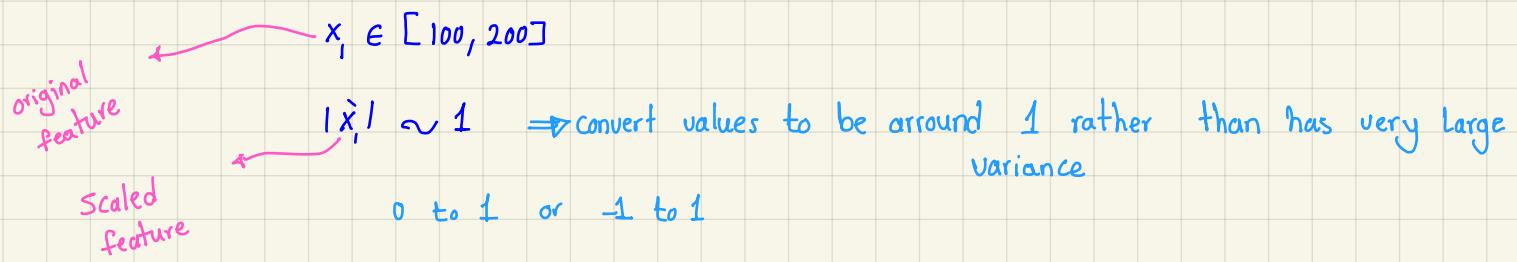
Note

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 a lot 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.



1 Min - Max Normalization:-

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

$$\text{so } x' \in [0, 1]$$

	x_i	x'_i
min	100	0
100	0	
150	0.5	
max	200	1
180	1.8	

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

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)

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

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

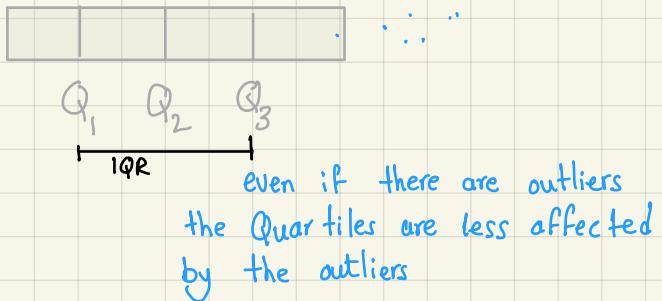
3 Quartiles

$$IQR = Q_3 - Q_1$$

↳ it's a Robust scaling.

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

IQR → InterQuartile Range



there are measures of spread :-

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

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

مقدار تباين 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 Naive bayes classifier or decision tree feature scaling is not important then

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

Variations of Batch gradient descent

"vanilla"

① \Rightarrow 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 a lot of memory 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 least 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 minima 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:- $\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla J(\theta)$

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

The equation of the momentum is:-

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

next step

to control the amount I will use of the old gradient descent

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

the θ_{temp} :

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

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

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

Nesterov Accelerated GD (NAG)

- In the standard momentum method:
 - first computes the gradient at the current position;
 - then takes a big jump in the direction of the accumulated gradient.
- In NAG:
 - first make a big jump in the direction of the previous accumulated gradient; (looking ahead step)
 - then measure the gradient where you end up and make a correction.

blue vectors = standard momentum
 brown vector = jump red vector = correction
 green vector = accumulated gradient

It is always better to correct a mistake right after you have made it.

- 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).

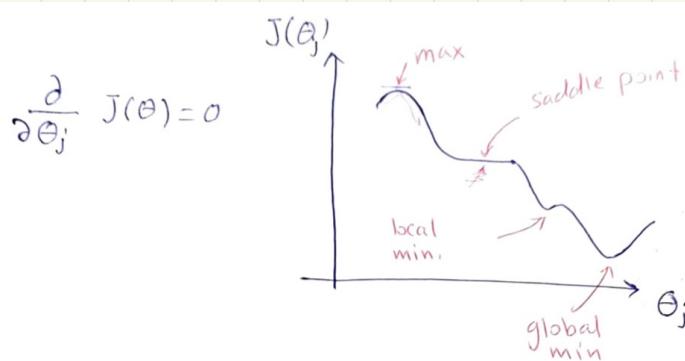
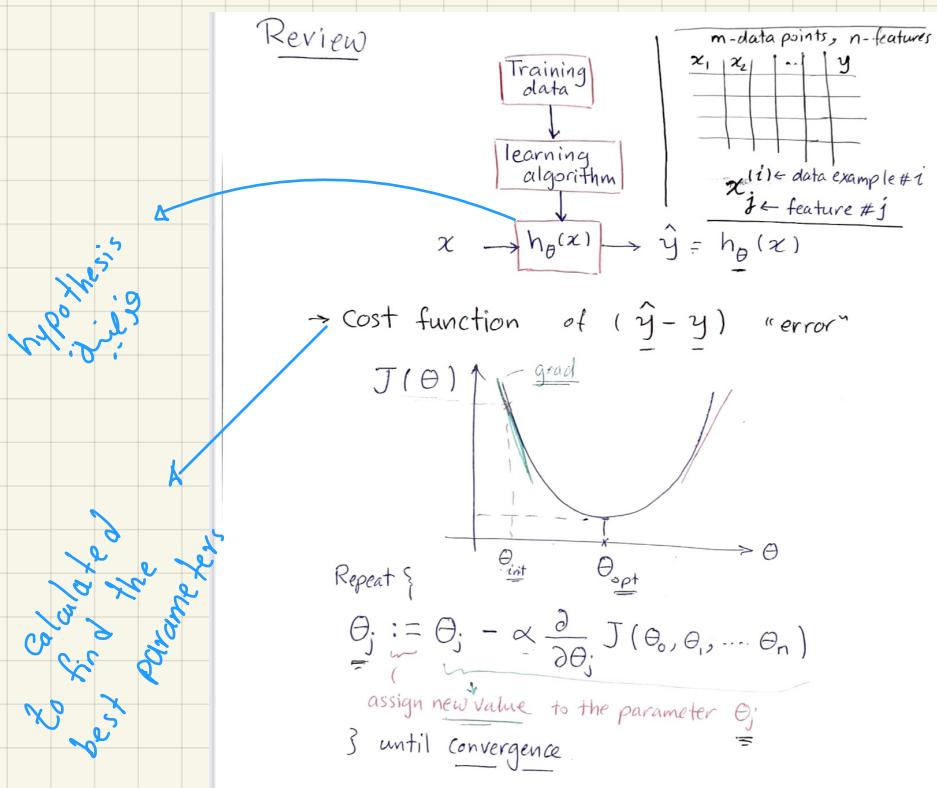
* it corrects the step size before taking the step \Rightarrow Nesterov

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

Lecture 4

$m \Rightarrow$ data points

$n \Rightarrow$ 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

Absorption 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 absorption effect the full rank matrix may not recognized as full rank because columns has this effect . \rightarrow 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 α although it's not actual

feature and it always = 1 to be able to belong to \mathbb{R}^{n+1}

as thetas

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

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

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

$$\vec{\theta}^{(+) \dagger} = \vec{\theta}^{(+) \ddagger} - \alpha \nabla (J(\vec{\theta}^{(+) \ddagger}))$$

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

different ways to write

the equations

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

for all data points m

it will not make difference

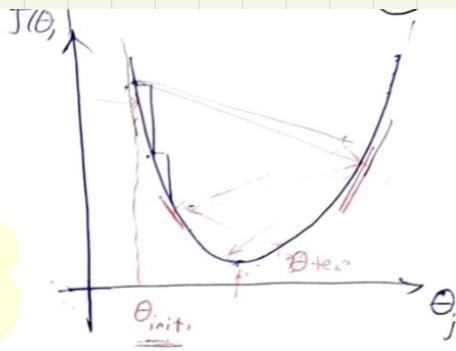
if $\hat{y} - y$ or $y - \hat{y}$ because we usually use $(\cdot)^2$ or absolute .

Momentum GD

$$v(t) =$$

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

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



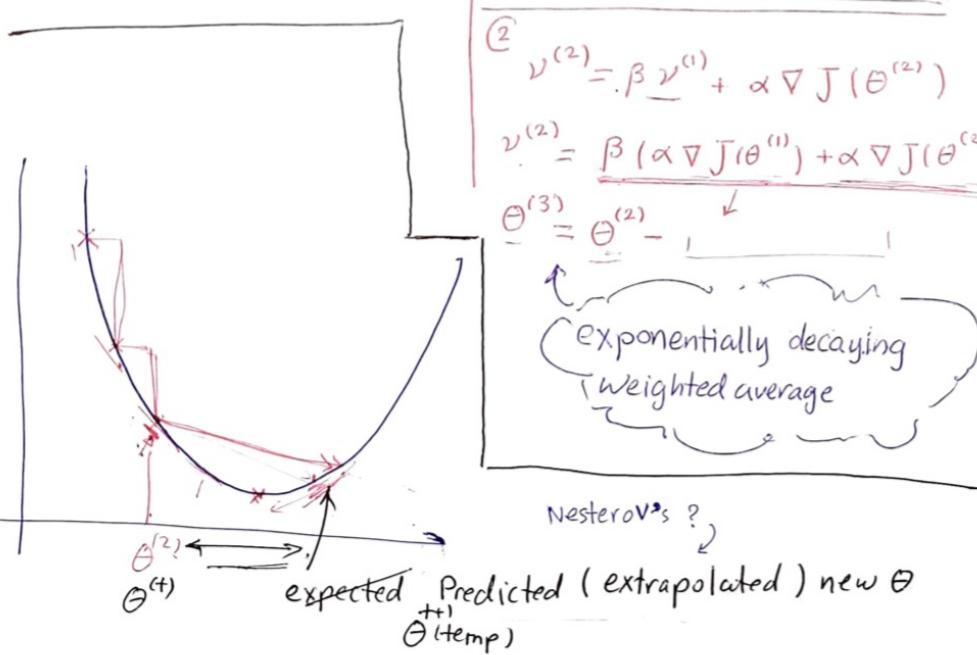
Equations

$$\begin{aligned} ① \quad \theta^{(0)} &= \theta_{\text{init}} \\ v^{(0)} &= 0 + \alpha \nabla J(\theta^{(0)}) \\ \theta^{(1)} &= \theta^{(0)} + v^{(0)} \\ \theta^{(2)} &= \theta^{(1)} - \alpha \nabla J(\theta^{(1)}) \end{aligned}$$

$$\begin{aligned} ② \quad v^{(2)} &= \beta v^{(1)} + \alpha \nabla J(\theta^{(2)}) \\ v^{(2)} &= \frac{\beta (\alpha \nabla J(\theta^{(1)})) + \alpha \nabla J(\theta^{(2)})}{\beta + 1} \\ \theta^{(3)} &= \theta^{(2)} - \frac{v^{(2)}}{\beta + 1} \end{aligned}$$

↑ exponentially decaying
↓ weighted average

Nesterov's ?



So that if they are in same 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 $\gamma v(t-1)$ becomes smaller and smaller i.e., this equation holds the farther updates by a small magnitude and recent updates by a large magnitude.

Example in slide 36 \Rightarrow Momentum based GD

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

↳ because $v_0 = 0$

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

→ 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 :-

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

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

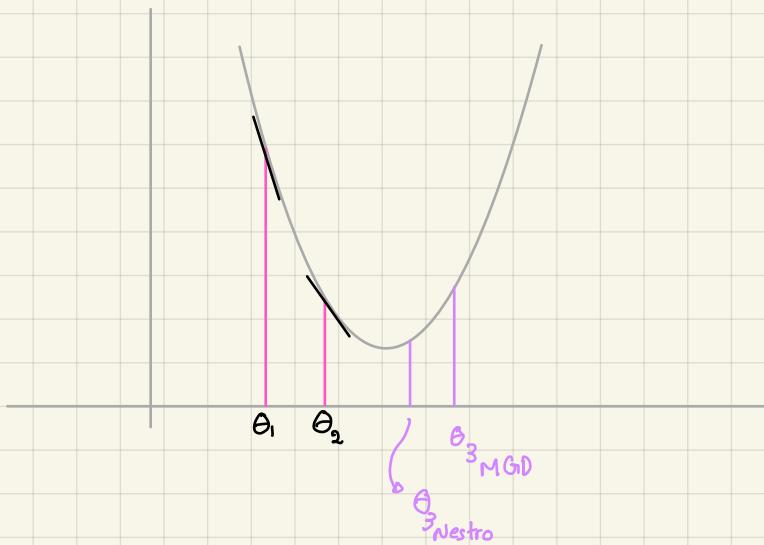
In Nesterov it will reduce the oscillation and prevent the overshoot of Momentum based

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 Nesterov method :-



Momentum

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

$$\Rightarrow v^{(t)} = \beta (\underbrace{\text{grad}(J(\theta_1))}_{-\text{ve}} + \underbrace{\text{grad}(J(\theta_2))}_{-\text{ve}})$$

$$\Rightarrow \theta_0 v^{(t)} = -\text{ve} \uparrow$$

$$\therefore \theta_3 = \theta_2 - (-\text{ve} \uparrow)$$

$$\theta_3 \gg \theta_2$$

Nesterov

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

$$\Rightarrow v^{(t)} = \beta (\underbrace{\text{grad}(J(\theta_1))}_{-\text{ve}} + \text{grad}(\theta_{\text{temp}}))$$

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

$$\theta_{\text{temp}} \gg \theta_2 \rightarrow \text{here comes the advantage of Nesterov}$$

$$\theta_3 = \theta_2 - (-\text{ve} + \text{ve})$$

So it will be reduce the effect

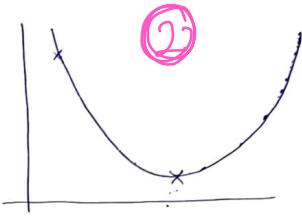
in θ_{temp} we update by θ_{new} not like the v we update by the old value

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)

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

↑ numerical stabilizer

if gradient = 0 to prevent divide by 0
expressed with velocity term $v_j^{(t+1)} = v_j^{(t)} + (\text{grad}(\theta_j))_j^2$

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

• 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

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

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

- Can we use different α for each parameter?

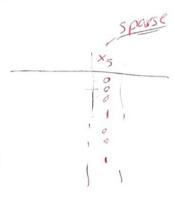
$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \vec{\alpha} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

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

for $\vec{\alpha}$

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

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



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

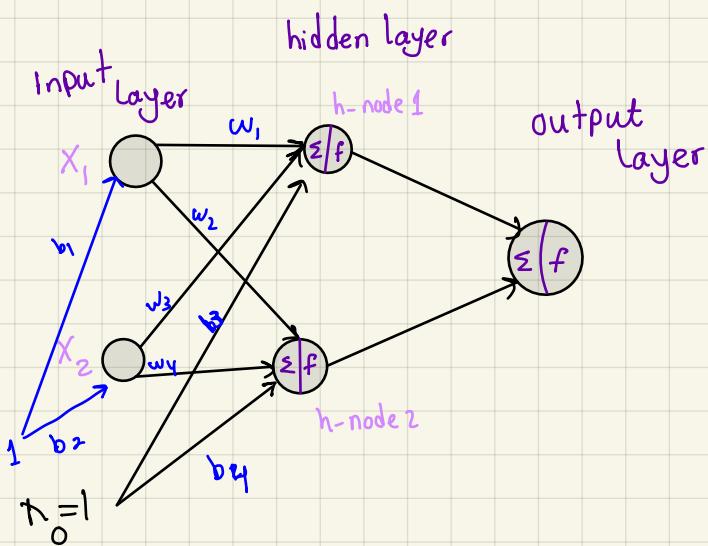
\Rightarrow RMS calculate gradient in the normal way no difference but it uses Adaptive learning with β

\Rightarrow Adam will work on gradient and Adapting Learning rate

will be discussed in next lecture

Lecture 5

Neural Network:-



Each node contains "perform" 2 things:-

① Σ "summation"

② $f \Rightarrow$ 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"
 \Rightarrow all weights are randomly initialized

at h_1 :-

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

$$\text{out}_{h_1} = f(z_{h_1}) = f(x_1 w_1 + x_2 w_3 + b_3)$$

$$f \text{ is } \rightarrow \text{sigmoid} \leftarrow \text{out}_{h_1} = \frac{1}{1 + e^{-z_{h_1}}} = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_3 + b_3)}}$$

$$\text{out}_{h_2} = f(x_1 w_2 + x_2 w_4 + b_4)$$

$$z_{o_1} = \text{out}_{h_1} w_5 + \text{out}_{h_2} w_6 + b_5$$

$$y = f(z_{o_1}) = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_3 + b_3)}}$$

we update all the weights

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

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

:

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

for example if I use MSE

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

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

$$\begin{aligned}\hat{y} &= f(z_{01}) = \frac{1}{1 + e^{-(\text{Out}_{h1}w_5 + \text{Out}_{h2}w_6 + b_5)}} \\ &= \frac{1}{1 + e^{-z_{01}}},\end{aligned}$$

$$\frac{\partial J()}{\partial w_1} = \frac{\partial J()}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_{01}} * \frac{\partial z_{01}}{\partial w_1} \rightarrow \text{chain Rule}$$

$$\frac{\partial J()}{\partial w_2} = \frac{\partial J()}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_{01}} * \frac{\partial z_{01}}{\partial w_2}$$

:

and so on

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

$$\text{So update } w_{1 \text{ new}} = w_{1 \text{ old}} - \alpha * 0.1 = w_{1 \text{ new}}$$



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 ⇒ ($t \Rightarrow$ high) & ($t-1 \Rightarrow$ less) & ($t-2 \Rightarrow$ less than $t-1$) ... etc

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

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

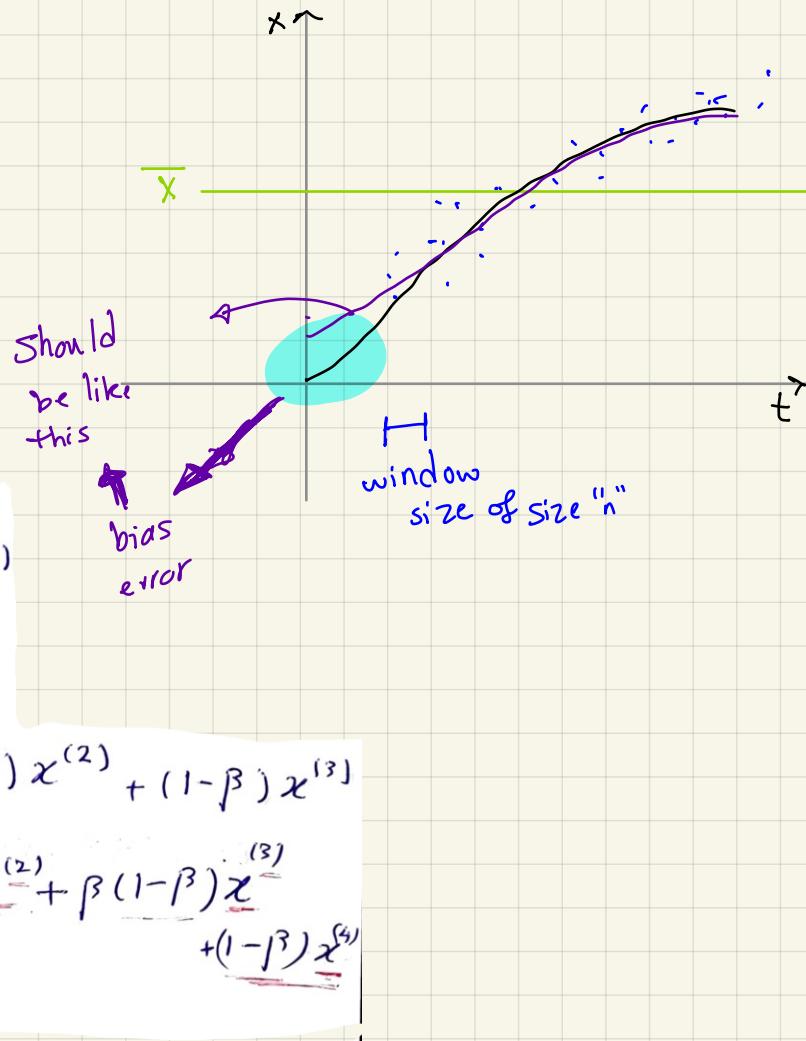
$$v^{(1)} = \beta v^{(0)} + (1-\beta)x^{(1)}$$

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

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

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

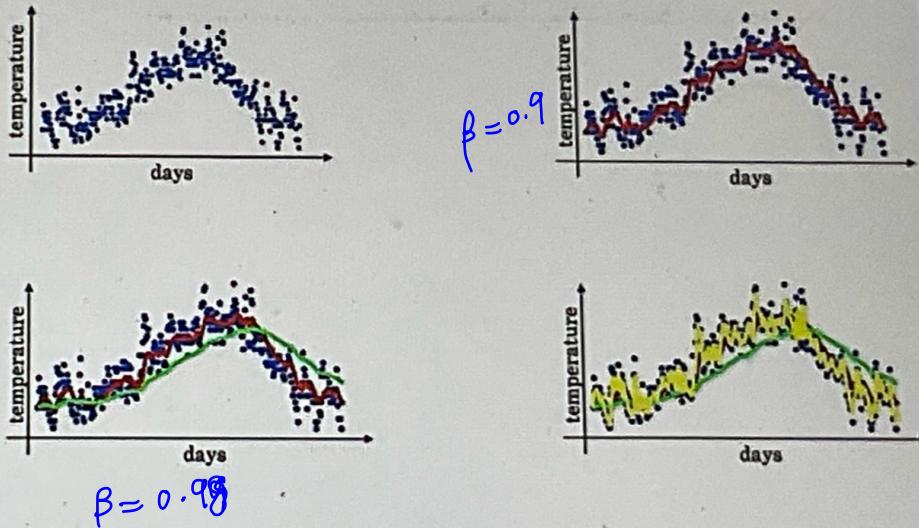
$$v^{(4)} = \beta^3(1-\beta)x^{(1)} + \beta^2(1-\beta)x^{(2)} + \beta(1-\beta)x^{(3)} + (1-\beta)x^{(4)}$$



→ 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
- $\beta=0.9$ averaging over 10 days
- $\beta=0.98$ averaging over 50 days
- $\beta=0.5$ averaging over 2 days



the green line is below the

data because the early points were small

and increased later because data in the middle are high

→ because $v_0 = 0$ a problem appears which is bias needs bias correction
it happens at the beginning of the curve "large error"

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

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

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

{

by time t will be very large

and \hat{v} will be very small

"eliminated"

Bias Correction

Bias correction

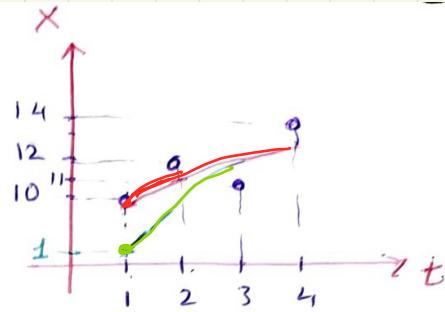
let $\beta = 0.9$

Without Bias correction

$$v^{(0)} = 0 \quad \leftarrow$$

$$\begin{aligned} v^{(1)} &= \beta v^{(0)} + (1-\beta) x^{(1)} \\ &= 0.9 \times 0 + 0.1 \times 10 \\ &= 1 \end{aligned}$$

$$\begin{aligned} v^{(2)} &= \beta v^{(1)} + 0.1 \times 12 \\ &= 1 + 1.2 \end{aligned}$$



With Bias corrected \leftarrow

$$\begin{aligned} \hat{v}^{(1)} &= \frac{v^{(1)}}{1-\beta^1} \\ &= \frac{1}{1-0.9^1} = \frac{1}{0.1} = 10 \end{aligned}$$

$$\begin{aligned} \hat{v}^{(5)} &= \frac{v^{(5)}}{1-\beta^5} \\ &\approx v^{(5)} \end{aligned}$$

for large time index.

$$1-\beta^t \approx 1$$

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

Momentum Review:-

Quick Note :- you might find some references write $v^{(t+1)} = \beta v^{(t-1)} + \alpha \nabla (\mathcal{J}(\theta^{(t)}))$

$$v^{(t)} = \beta v^{(t-1)} + \alpha \nabla (\mathcal{J}(\theta^{(t)})) \quad \xrightarrow{\hspace{1cm}} I$$

$$\theta^{(t+1)} = \theta^{(t)} - v^{(t)} \quad \xrightarrow{\hspace{1cm}} II$$

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

No problem

from I & II

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

EWMA Form:-

may be defined as:-

velocity, $v^{(t)}$

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

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

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

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

⇒ 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.

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

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

Adam (Adaptive Momentum)

EWMA momentum

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

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

RMS prop.

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

↑ bias-corrected
bias-corrected
 $v_2^{(t)}$ $\hat{v}_1^{(t)}$

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

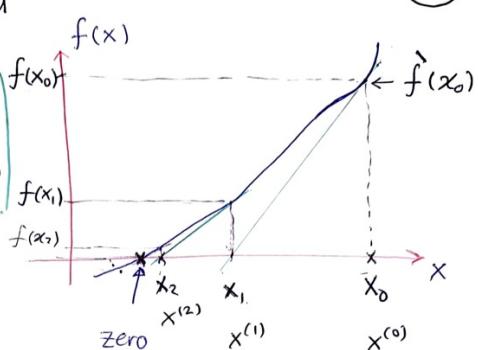
value of x that $f(x) = 0$

Newton's Method

(7)

Zero of a function;

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



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

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

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

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

$$x^{(+)} = x^{(+-)} - \frac{f(x^{(+-)})}{f'(x^{(+-)})}$$

the derivative = zero in 3 cases

— max point

— min //

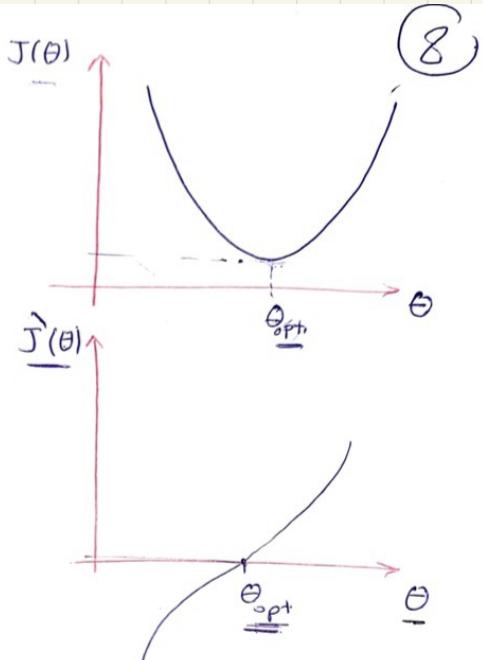
— saddle //

Zero of $J'(\theta)$

that's the
problem of
Newton's method

\Rightarrow minimum of $J(\theta)$
(or maximum, or
saddle point, or
local minimum!)

→ 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 with the H matrix

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(n³)

Remember

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

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

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

$$H = \begin{bmatrix} H_{i,j} \end{bmatrix}$$
$$H_{i,j} = \frac{\partial^2 f}{\partial \theta_i \partial \theta_j}$$

$\nabla J(\theta_1, \dots, \theta_n) = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix}$

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

→ for further search

Newton's method advantages:-

- doesn't need learning rate \Rightarrow 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 \rightarrow take small step
 - If curve is flat \rightarrow 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)

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 parameters are updated in the same manner.

In GD we linearly approximate

Summary

الخطوات المتبعة

$$\text{Min max} \quad \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

$$\text{mean} \quad \frac{x - \bar{x}}{\sigma}$$

$$\text{IQR} \quad \frac{x - Q_2}{\text{IQR}}$$

$$\nabla = \frac{\partial f}{\partial x}$$

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

Standard GD

Momentum based GD

The equation of the momentum is:-

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

to control the amount I will use of the old gradient descent

Nesterov next step

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

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

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

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

Nesterov Accelerated GD (NAG)

- In the standard momentum method:
 - first computes the gradient at the current position;
 - then takes a big jump in the direction of the accumulated gradient.
- In NAG:
 - first make a big jump in the direction of the previous accumulated gradient; (looking ahead step)
 - then measure the gradient where you end up and make a correction.

It is always better to correct a mistake right after you have made it.

blue vectors = standard momentum
brown vector = jump
red vector = correction
green vector = accumulated gradient

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

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

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

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

different ways to write
the equations

Momentum GD

$$v(t) =$$

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

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

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

Example in slide 36 \Rightarrow Momentum based GD

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

because $v_0 = 0$

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

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

$$\begin{aligned} \theta^{(t+1)} &= \theta^{(t)} - v^{(t)} \\ \theta^{(t+1)} &= \theta^{(t)} - \underbrace{\beta v^{(t-1)}}_{\theta_{\text{temp}}} - \alpha \nabla J(\theta_{\text{temp}}) \end{aligned}$$

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

The momentum based was :-

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

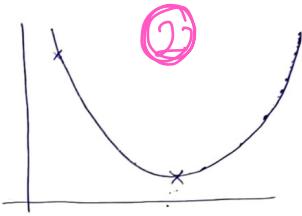
$$\theta^{(t+1)} = \theta^{(t)} - v^{(t)}$$

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)

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

↑ numerical stabilizer

if gradient = 0 to prevent divide by 0

$$\text{expressed with velocity term } v_j^{(t+1)} = v_j^{(t)} + (\text{grad}(\theta_j))_j^2$$

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

• 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

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

- Can we use different α for each parameter?

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

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

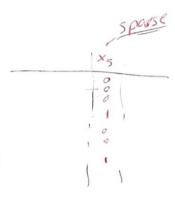
for $\vec{\alpha}$

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

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

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
- β (beta) is used to control the amount of history

Adagrad

Comparison

Adagrad, RMS Prop, Adam

$$v^{(t-1)} = v^t + \nabla^2$$

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

RMS Prop

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

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

→ Exponentially decaying weighted sum is a special case of Exponentially weighted moving average

Bias Correction

Bias correction

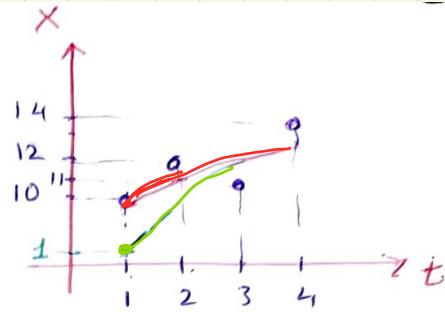
let $\beta = 0.9$

Without Bias correction

$$v^{(0)} = 0 \quad \leftarrow$$

$$\begin{aligned} v^{(1)} &= \beta v^{(0)} + (1-\beta) x^{(1)} \\ &= 0.9 \times 0 + 0.1 \times 10 \\ &= 1 \end{aligned}$$

$$\begin{aligned} v^{(2)} &= \beta v^{(1)} + 0.1 \times 12 \\ &= 1 + 1.2 \end{aligned}$$



With Bias corrected \leftarrow

$$\begin{aligned} \hat{v}^{(1)} &= \frac{v^{(1)}}{1-\beta^1} \\ &= \frac{1}{1-0.9^1} = \frac{1}{0.1} = 10 \end{aligned}$$

$$\begin{aligned} \hat{v}^{(5)} &= \frac{v^{(5)}}{1-\beta^5} \\ &\approx v^{(5)} \end{aligned}$$

for large time index.

$$1-\beta^t \approx 1$$

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