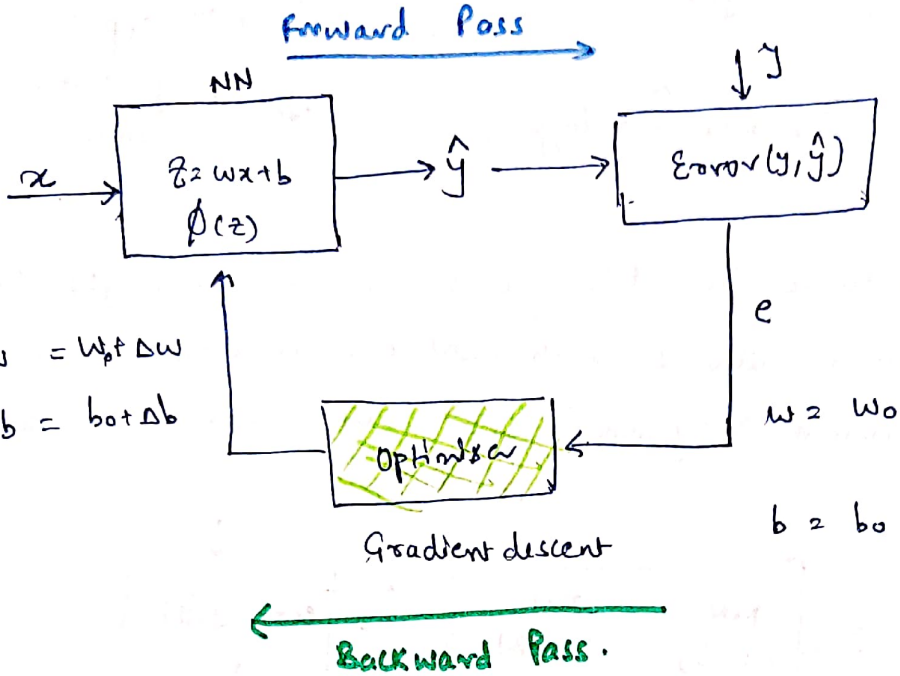
Optimisers



$$w = w_0 - \eta \frac{\partial e}{\partial w}$$

$$b = b_0 - \eta \frac{\partial e}{\partial b}$$

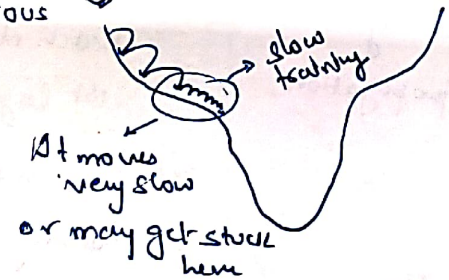
Drawback of Gradient Descent

$$\Delta w \propto \left.\frac{\partial C}{\partial w}\right|_{w=w_{old}}$$

↓

It does not get influenced by previous updates.

hence this is only dependent on current weight update



Observation

(i) if $\Delta w \uparrow\uparrow$ if $\frac{\partial C}{\partial w} \uparrow\uparrow$

if slope steep at that point weight update will be high

$\Delta w \downarrow\downarrow$ if $\frac{\partial C}{\partial w} \downarrow\downarrow$ → flat surface

Simplified repr of gradient descent

$$w_{new} = w_{old} - \eta \left.\frac{\partial C}{\partial w}\right|_{w=w_{old}}$$

or

$$\theta_{new} = \theta_{old} - \eta \nabla_\theta J(\theta)$$

↑ Cost fn.

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ b_1 \\ \vdots \\ b_2 \end{bmatrix} = \begin{bmatrix} w_{1old} \\ \vdots \\ b_{1old} \\ \vdots \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial b_1} \\ \vdots \end{bmatrix}$$

$$\Delta w = -\eta \frac{\partial C}{\partial w}$$

$$\Delta w \propto \frac{\partial C}{\partial w} \qquad \nabla\theta \propto \nabla_\theta J(\theta)$$

↳ slope of tangent

(iii) chain rule at lower layers includes multiple $\frac{\partial C}{\partial w}$ terms and hence (2)

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial a_2} \cdot \frac{\partial a_L}{\partial z_L} \cdot \frac{\partial z_L}{\partial w_2} \Big] \text{ lower weight update}$$

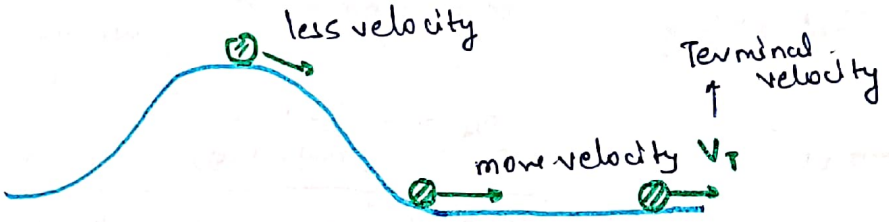
This will train very slowly at lower layers as compared to first one

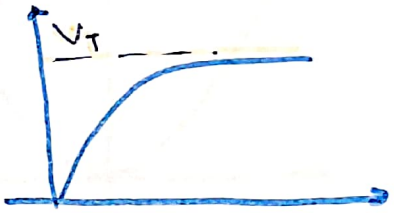
Type of Optimisers : Fast Optimiser

1) Momentum Optimisation
2) Nesterov accelerator gradient
3) Adaptive Gradient
4) RMS Prop
5) Adam Optimisation.

Momentum Optimisation (Boris Polyale) 1964

Idea





It tries to accumulate past gradient as well so that you reach minima faster.

Algorithm

$m_{new} \leftarrow \beta m_{old} + \eta \nabla_\theta J(\theta)$

$\theta \leftarrow \theta - m$

or

$m_{new} = \beta m_{old} + \eta \frac{\partial C}{\partial w}\Big|_{w = w_{current}}$

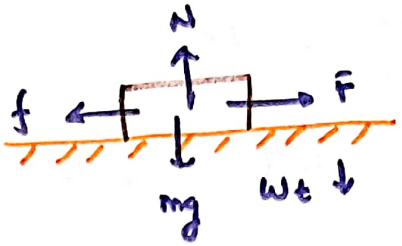
$w_{new} = w_{old} - m$

↳ bias update will also happen in same way

* $\theta = \theta - [\beta m + \eta \nabla_\theta J(\theta)]$

β = coefficient of momentum analogous to coefficient of friction.

$\beta = 0$ $\Rightarrow$ frictionless surface $\quad \mu = 0$, $f = 0$



$f = \mu N = \mu mg$

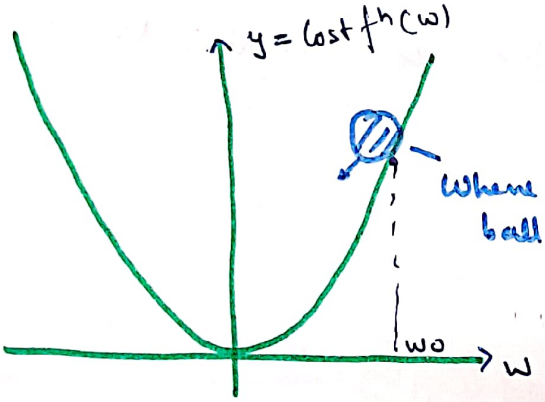
$m = \cancel{\beta m^{\nearrow 0}} + \eta \left.\frac{\partial c}{\partial w}\right|_{w = \text{current weight}}$

$= \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_{current}}$

$$\boxed{w = w - m = w - \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_{current}}}$$

→ This is normal gradient descent if $\beta = 0$

What is value of β? It is experimentally proven in most of the cases that $\beta = 0.9$ but you can take any other value as well and not bounded by it.



Step ~~Case~~ 1 :-

Initially $w = w_0$, $m_0 = 0$, $\beta = 0.9$

$m_1 = \cancel{\beta m_0^{\nearrow 0}} + \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_0}$

$m_1 = \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_0}$

$$\boxed{w_1 = w_0 - m_1 = w_0 - \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_0}}$$

like a gradient descent

Step 2 :-

$m_1 = \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_0}$, $\beta = 0.9$, $w = w_1$

$m_2 = \beta m_1 + \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_1}$

$m_2 = \left(\beta \eta \left.\frac{\partial c}{\partial w}\right|_{w = 0}\right) + \eta \left.\frac{\partial c}{\partial w}\right|_{w = w_1}$

✓ It considers past weight updates and does not solely depends on the current weights.

Step n :

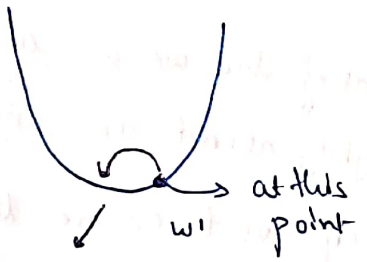
$$m_n = \eta \left[\beta^{n-1} \frac{\partial c}{\partial w}\Big|_{w=w_0} + \beta^{n-2} \frac{\partial c}{\partial w}\Big|_{w=w_1} + \cdots \frac{\partial c}{\partial w}\Big|_{w=w_{n-1}} \right]$$

It takes more fraction of latest one & less fraction of previous one.

Drawback

① One extra parameter β but $\beta = 0.9$ works well for most of the cases.

②



$$\eta \left[\beta^{n-1} \frac{\partial c}{\partial w}\Big|_{w=w_0} + \cdots \frac{\partial c}{\partial w}\Big|_{w=w'} \right]$$

↓

This value is already very less

Oscillation Issue

It oscillates closer to global or local minima due to accumulation of past momentum

Hence in this case past gradient will dominate over the current gradient

Advantages

① Fast Conversion

② Oscillation can also help to come out of local minima

Keras

optimizer = tf.keras.optimizers.SGD(lr = 0.01, momentum = 0.9)

Ada Grad Adaptive Gradient

Nesterov Accelerated Gradient (NAG) - Yuri Nesterov, 1983

aka Nesterov Momentum Optimisation as it's faster than momentum optimisation

What is new?

It calculates gradient slightly ahead in the direction of momentum. Instead of calculating at θ but $\theta - \beta m$ or $w - \beta m$.

Algorithm

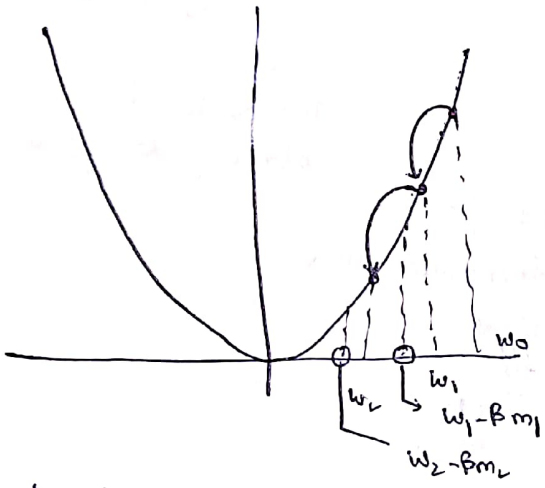
$$m = \beta m + \eta \frac{\partial c}{\partial w}\Big|_{w=(w-\beta m)}$$

$$w = w - m$$

or

$$m \leftarrow \beta m + \eta \nabla_\theta (\theta - \beta m)$$

$$\theta \leftarrow \theta - m$$



It is also accumulating but we are also looking for terms slightly ahead and knowing what exists. This helps us to ensure that we are actually converging faster and also know what exists ahead.

Advantages

1. Faster than momentum optimisation, it gives us less oscillations. Oscillations are significantly reduced at the minimum.
2. Reaches close to minima because of less oscillations.

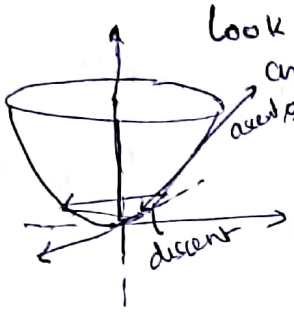
Disadvantages

1. Extra parameter to tune i.e β but as you know $\beta = 0.9$ works well in most of the cases.

* optm. = tf.keras.optimizers.SGD(lr= , momentum = 0.9, nesterov = True)

Ada Grad Adaptive Gradient

It solves one of the problems of gradient descent or the past fast optimisers that we have seen. → The problem is known as. **Elongated Bowl Problem**

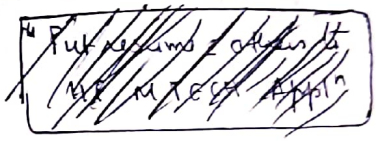


$$w = w - \eta \frac{\partial c}{\partial w}$$

$$w = w - \nabla C$$

↳ this negative sign leads to gradient descent

Sometimes this negative descent in an elongated bowl may lead us to a divergence track rather than convergence and this particular problem is solved by ada grad.



The $-\nabla C$ does not always focus towards the direction of global minimum.

Dot product :- $\vec{v} = x\hat{i} + y\hat{j}$ $\vec{p} = a\hat{i} + b\hat{j}$

Scaler Product $\vec{v} \cdot \vec{p} = xa + yb = |\vec{v}||\vec{p}| \cos\theta$

Algorithm (Adaptive Gradient)

* It corrects the direction in the initial steps. → scaler term

$$s \leftarrow s + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s + \varepsilon}$$

or

$$s = s + \left(\frac{\partial c}{\partial w}\Big|_w\right)^2$$

$$w = w - \eta \left(\frac{\partial c}{\partial w}\right)_w \Big/ \sqrt{s + \epsilon}$$

s = scaling down factor

ϵ = Smoothing term to avoid zero division error = 10^{-7}

$\otimes$ = Element wise multiplication . dot product

$\oslash$ = element wise division

$$s \leftarrow s + \frac{\nabla_\theta J(\theta)}{\vec{v}} \otimes \frac{\nabla_\theta J(\theta)}{\vec{v}}$$

$$\theta \leftarrow \theta - \eta \left(\frac{\nabla_\theta J(\theta)}{\sqrt{\epsilon + |\vec{v}|^2 + \epsilon}} \right) \rightarrow \frac{\vec{v}}{\sqrt{s_0 + |\vec{v}|^2 + \epsilon}}$$

* if $s_0 = 0$ $\epsilon \approx 0$ (1st iteration)

$$\frac{\vec{v}}{\sqrt{|\vec{v}|^2}} = \hat{v} \text{ unit vector}$$

* After some iteration

$$\frac{\vec{v}}{\sqrt{s + |\vec{v}|^2 + \epsilon}} = \vec{v}'$$

$|\vec{v}'| < \hat{v}$ as $\vec{v}$ is divided by an increased denominator.

Step by step

Step 1: Initially $s_0 = 0$, $w = w_0$

$$s_1 = \cancel{s_0}^{0} + \left(\frac{\partial c}{\partial w}\Big|_{w=w_0} \right)^2 \quad -(1)$$

$$w_1 = w_0 - \eta \frac{\left(\partial c / \partial w |_{w=w_0} \right)}{\sqrt{\left(\frac{\partial c}{\partial w}\Big|_{w=w_0} \right)^2 + \epsilon}} \quad -(2)$$

$\epsilon \approx 0$

$$w_1 = w_0 - \eta \left(\frac{\left(\partial c / \partial w |_{w=w_0} \right)}{\left| \frac{\partial c}{\partial w}\Big|_{w=w_0} \right|} \right)$$

$$w_1 = w_0 - \eta \hat{v}$$

$\therefore \eta \in (0,1) \Rightarrow$ overall weight update will be very less.

step 2:

(8)

$$s_2 = s_1 + \left(\frac{\partial c}{\partial w}\Big|_{w=w_1}\right)^2$$

$$s_2 = \underbrace{\left(\frac{\partial c}{\partial w}\Big|_{w=w_0}\right)^2}_{\text{past}} + \underbrace{\left(\frac{\partial c}{\partial w}\Big|_{w=w_1}\right)^2}_{\text{present}}$$

Now we are taking 100% of it unlike the previous cases.

$$w_2 = w_1 - \eta \frac{\frac{\partial c}{\partial w}\Big|_{w=w_1}}{\sqrt{s_2 + \epsilon}}$$

larger term than the previous one

step n:

$$s_n = \sum_{i=0}^{n} \left(\frac{\partial c}{\partial w}\Big|_{w=w_i}\right)^2$$

s_n is very large due to accumulation compared to s_1

$$w_{n+1} = w_n - \eta \frac{\frac{\partial c}{\partial w}\Big|_{w=w_n}}{\sqrt{s_n + \epsilon}}$$

$$\Delta w \propto \frac{\frac{\partial c}{\partial w}\Big|_{w=w_n}}{\sqrt{\sum_{i=0}^{n} \frac{\partial c}{\partial w}\Big|_{w=w_i} + \epsilon}}$$

assumption $K = \left\{\sqrt{\sum_{i=0}^{n} \frac{\partial c}{\partial w}\Big|_{w=w_i} + \epsilon}\right\}$

$$\Rightarrow \Delta w \propto \frac{\frac{\partial c}{\partial w}\Big|_{w=w_n}}{K}$$

$$w = w - \eta \frac{\frac{\partial c}{\partial w}\Big|_{w=w_n}}{K}$$

$$\left(w = w - \frac{\eta}{K} \frac{\partial c}{\partial w}\Big|_{w=w_n}\right)$$

varying term

overall it is similar to gradient descent with learning rate

$\frac{\eta}{K}$ → decaying learning rate

K will keep increasing

This is the property of Adagrad

(9) Hence due to ada grad the jump will be lower and zig-zag fashion can be avoided. It is trying to self correct the direction

Gradient Descent	Ada Grad
- Moves quickly in direction of deepest descent without considering where global minima is located.	- It corrects its direction by scaling down the gradient vector along the steepest direction.

Advantages of Ada grad

1) Corrects the direction ~~initially~~ of gradient vector initially unlike the gradient descent.

2) Less tuning of learning rate is required because of decaying factor.

Disadvantages

1) ~~If~~ It often stops early before reaching global minima

2) It takes longer time to converge due to decaying learning rate.

** Even though Ada grad has direction correction advantage but due to point (1) in disadvantage it is not recommended to use

RMS Prop

↳ Geoffrey Hinton et. al.

Tackles the drawbeg of Ada grad. - early stopping problem. by accumulating gradient from recent iteration and using exponential decay

Algorithm

$$s \leftarrow \beta s + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$
$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s+\epsilon}$$

or

$$s = \beta s + (1-\beta)\left(\frac{\partial c}{\partial w}\Big|_{w_j}\right)^2$$
$$w = w - \eta\left(\frac{\partial c}{\partial w}\Big|_{w}\right) \Big/ \sqrt{s+\epsilon}$$

Smoothing term

β = Coefficient — value = 0.9

experimentally proven not any theoritical basis.

RMS Prop

$$w_1 = w_0 - \eta \frac{\frac{\partial c}{\partial w}\Big|_{w=w_0}}{\sqrt{\left(\frac{\partial c}{\partial w}\Big|_{w=w_0}\right)^2 (1-\beta) + \epsilon}}$$

(1−β) → 0.1

Smaller denominator

Ada Grad

$$w_1 = w_0 - \eta \frac{\frac{\partial c}{\partial w}\Big|_{w=w_0}}{\sqrt{\left(\frac{\partial c}{\partial w}\Big|_{w=0}\right)^2 + \epsilon}}$$

large denominator

$\Delta w_{RMS} > \Delta w_{Adagrad}$

* decay in learning rate is also low.

Keras

```
tf.keras.optimizers.RMSProp(learning_rate=0.01, rho=0.9)
```

* **Adams** → Adaptive momentum estimations.
  - It combines the idea of
    - Momentum optimisation
    - RMS prop
    - Exponential decay

Algorithm

1. $m \leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta J(\theta)$
2. $s \leftarrow \beta_2 s + (1-\beta_2)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
3. $\hat{m} \leftarrow \frac{m}{1-\beta_1^t}$ → iteration step.
4. $\hat{s} \leftarrow \frac{s}{1-\beta_2^t}$
5. $\theta \leftarrow \theta - \eta \hat{m} \oslash \sqrt{s+\epsilon}$

—or—

1. $m = \beta_1 m + (1-\beta_1) \frac{\partial c}{\partial w}\Big|_{w=w}$ → Inspired from momentum optimisation
   ↳ Exponential decay of gradients.
   β^3 β^2 β
   0.729 0.81 0.9
2. $s = \beta_2 s + (1-\beta_2)\left(\frac{\partial c}{\partial w}\Big|_w\right)^2$ → Inspired from RMS prop - Exponential decay of squared gradients
3. $\hat{m} = \frac{m}{1-\beta_1^t}$
4. $\hat{s} = \frac{s}{1-\beta_2^t}$
   → Bias correction step :- zero bias correction step
   ↳ Inspired from Time series smoothing operation
5. $w = w - \eta \frac{\hat{m}}{\sqrt{\hat{s}+\epsilon}}$

Step ③ & step ④

Initial condn :- $m_0 = 0$, $s_0 = 0$, $t_0 = 0$

$\beta_1 = 0.9$, $\beta_2 = 0.999$ $\epsilon = 10^{-8}$

$m_1 = \beta_1 m_0 + (1-\beta_1) \frac{\partial c}{\partial w}\Big|_{w=w_0}$ $\quad s_1 = \beta_2 s_0 + (1-\beta_2)\left(\frac{\partial c}{\partial w}\Big|_w\right)^2$

$m_1 = (1-\beta_1) \frac{\partial c}{\partial w}\Big|_{w=w_0}$ $\quad s_1 = (1-\beta_2)\left(\frac{\partial c}{\partial w}\Big|_w\right)^2$

Let's assume $\frac{\partial c}{\partial w}\Big|_{w=0} = 10$

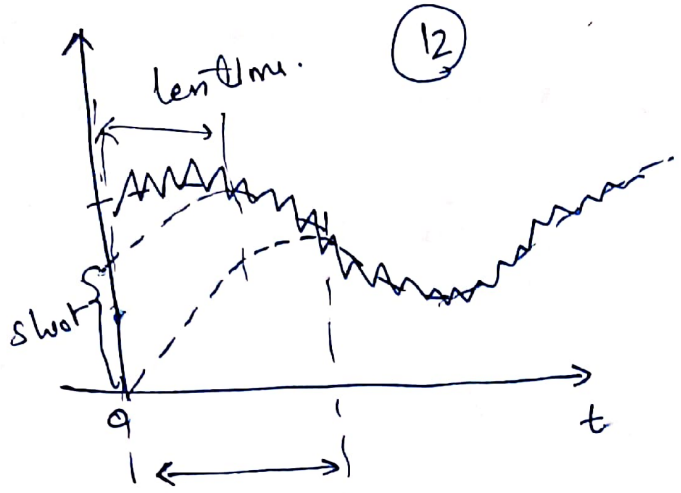
$m_1 = (1-0.9) \times 10 = 1$

$s_1 = (0.001) \times 100$
$= 0.1$

m_1 and s_1 both are close to zero

$\hat{m}_1 = \frac{m_1}{1-\beta_1^t} = \frac{m_1}{1-(0.9)^1} = \frac{1}{0.1} = 10$

$\hat{s}_1 = \frac{s_1}{1-\beta_1^t} = \frac{0.1}{0.001} = 100$



time taken to reach the following graph

In this example of time series we need to create a smoothing function. $\hat{m}_1$ & $\hat{s}_1$ shoot it far away from zero so that θ_t is able to follow along the graph pin very few iterations

* This is why we use a zero bias correction step.

Observation:- Since, we are starting m_0 and so from zero due to this pin initial few steps they will be biased towards zero. So to avoid that we are shooting θ_t away from zero using zero bias correction step.

Expanding Adam Algorithm.

$$\hat{m}_1 = \frac{m_1}{1-\beta_1^1} = \frac{(1-\beta_1)\frac{\partial C}{\partial w}\Big|_{w=w_0}}{(1-\beta_1)} = \frac{\partial C}{\partial w}\Big|_{w=w_0}$$

$$\hat{s} = \frac{s_1}{1-\beta_2^1} = \frac{(1-\beta_2)}{(1-\beta_2)}\left(\frac{\partial C}{\partial w}\Big|_{w=w_0}\right)^2 = \left(\left(\frac{\partial C}{\partial w}\right)\Big|_{w=w_0}\right)^2$$

$$w_1 = w_0 - \eta \frac{\frac{\partial C}{\partial w}\Big|_{w=w_0}}{\sqrt{\left(\frac{\partial C}{\partial w}\Big|_{w=w_0}\right)^2 + \epsilon}}$$ like Adagrad.

Step 2

$t_2 = t_1 + 1 = 2 \quad w_2 = w_1 \quad , \beta_1 = 0.9 \quad , \beta_2 = 0.999$

$$w_2 = w_1 - \eta \frac{\hat{m}_2}{\sqrt{\hat{s}_2 + \epsilon}}$$

$$\Delta w = \frac{\eta}{K} \hat{m}_2 = \eta' \hat{m}_2$$

K → varying number

η' → varying learning rate.

Q. When to use what algorithm ?

Class of Algorithm		Convergence speed	Convergent quality
Good for Simple ML data-set	SGD	2	3
	Momentum Opⁿ	2	3
	NAG	2	3
Not recommended to use	Ada Grad	2	1 (stops early)
	RMS Prop	3	2-3
	Adam	3	2-3

Adam → Best optimiser in terms of convergence speed and quality. Most commonly used optimiser.

RMS Prop → good for complex datasets. like images, video, etc.