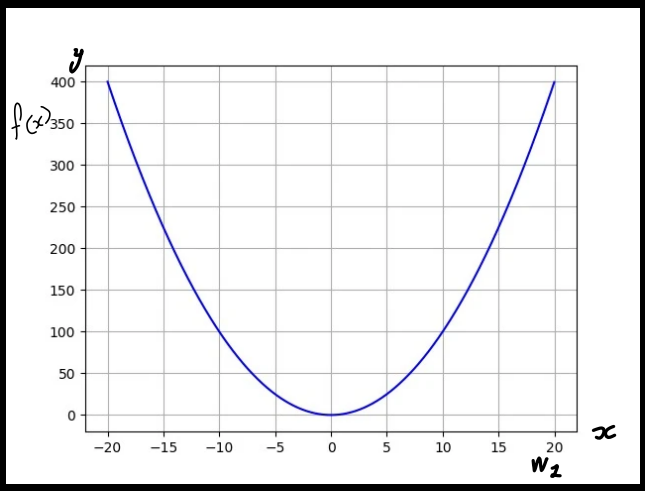
SGD with Momentum



→ How loss function is changing as a single weight is getting changed.

→ x axis = w_1, y axis = f(x)

↳ Loss = f(x) = $f(w_1)$

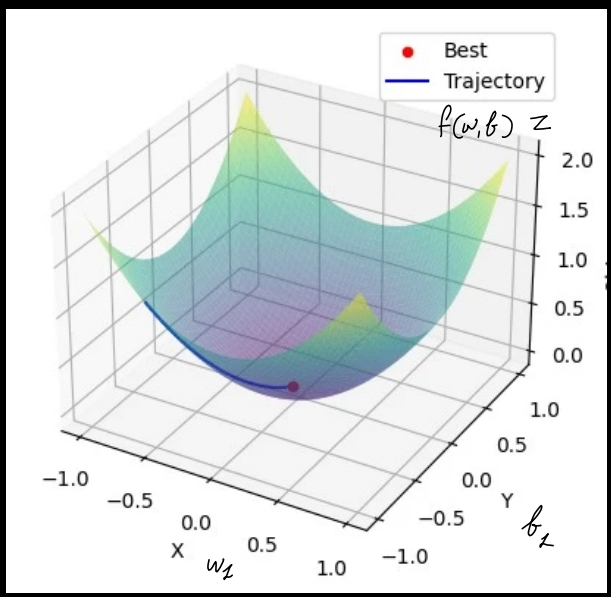
→ Our aim is to minimize loss as much as possible, by finding the optimal value of w_1

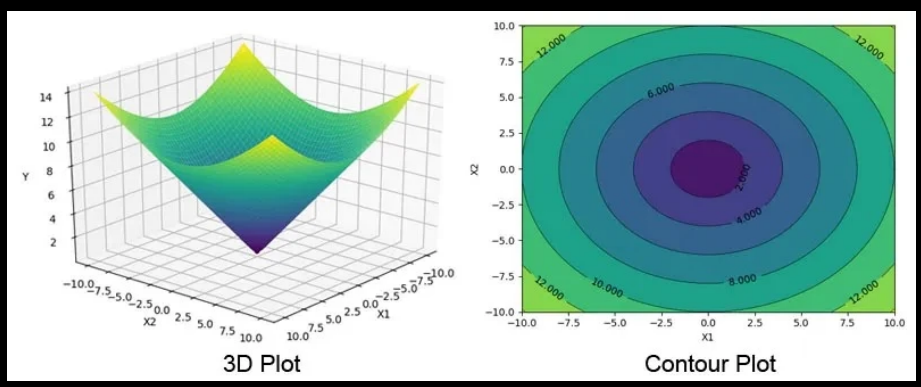
→ This is 3D representation of loss function W.R.T weight (w) & bias (b).

→ x axis = w, y axis = b, z axis = loss function = f(w,b)

→ How loss is changing with respect to 'w' & 'b'.

→ Our aim is to find such optimal value of w & b which brings down loss function as much as possible.



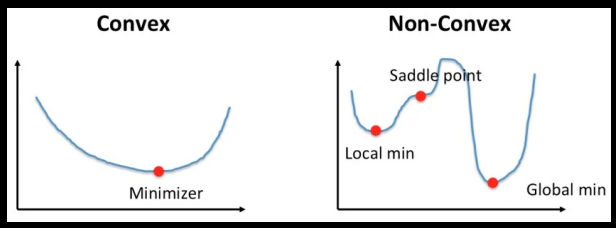


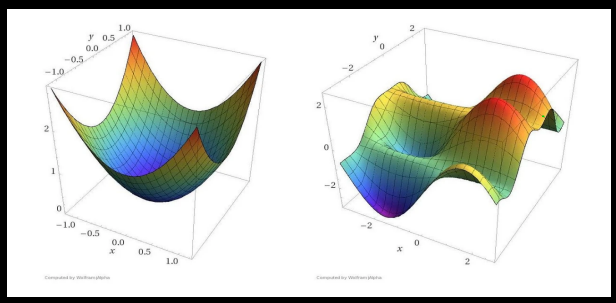
→ Contour plot is a simple 2D visulization of 3D plot from the top angle.

→ Contour plot shows lost dimension with the help of colour.

→ As given image shows, blue region shows down side.

Convex vs Non-convex optimization

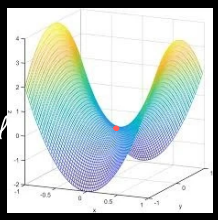




Problems in Non-convex opt.

→ local minima trap

→ Saddle point



↳ in one direction, slope is upward and another direction, slope is downwards for long distance.

→ High curvature

↳ Small circle = Small radius = High curvature
↳ Big circle = large radius = low curvature

Momentum Optimization

Target Problems

- High curvature
- Consistent gradient.
- Noisy gradient. (local minima trap)

Basic idea

→ It accelerates the convergence of gradient descent in that direction which previous gradient strongly suggests.

→ If our gradients are sending us in one direction then we will accelerate the convergence speed.

Single most benefit of momentum optimization

→ Speed

Mathematics

$$W_{t+1} = W_t - \alpha \Delta W_t$$

if V = Velocity then, now

$$W_{t+1} = W_t - V_t$$ where

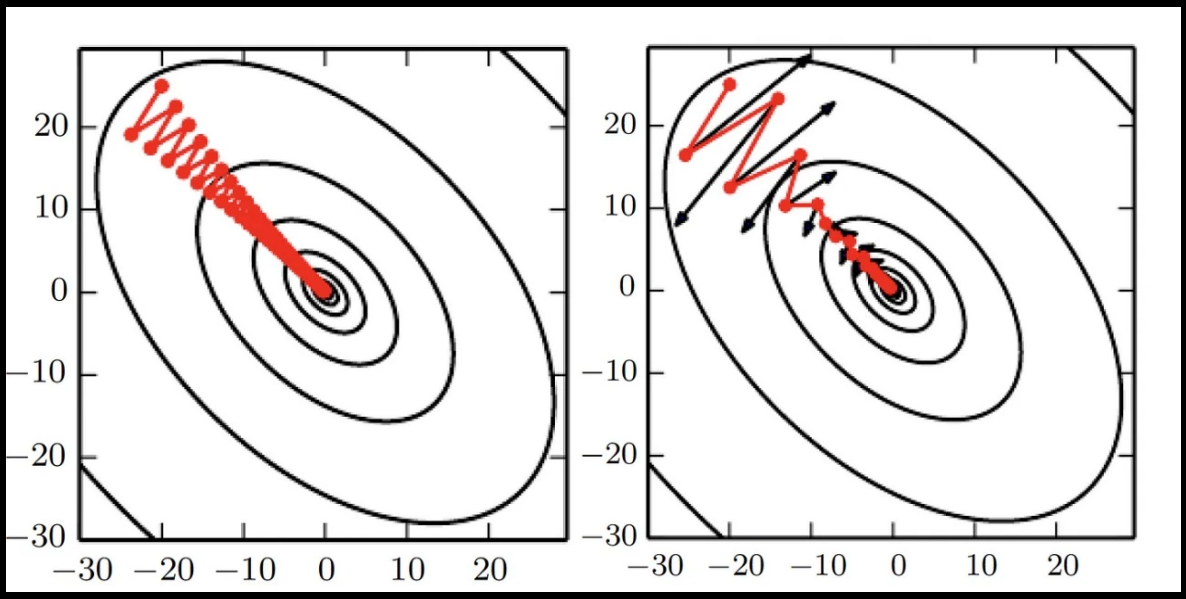
$$V_t = \underbrace{\beta V_{t-1}}_{\text{(past velocity) mo...um}} + \alpha \nabla W_t$$

β = 0 to 1

→ We will use all our past velocity as momentum and that will provide acceleration.

SGD

SGD with momentum



Role of β in momentum.

$$W_{t+1} = W_t - V_t$$

$$V_t = \beta V_{t-1} + \alpha \nabla W_t$$

if $\beta = 0$ (← decay) then

$$V_t = \alpha \nabla W_t$$

$$W_{t+1} = W_t - V_t$$

$$= \boxed{W_t - \alpha \nabla W_t} = \text{SGD}$$

→ Hence if $\beta = 0$, momentum will act like SGD.

→ Here, β is a kind of decaying factor

→ Newer velocity will have more impact than older.

→ Even $\beta = 1$ won't perform as expected, hence β should be 0.9 or 0.99...

(No decay)

Nesterov Accelerated Gradient (NAG)

-> NAG is a different technique to generate momentum.
-> NAG is a technique in which we reduces oscillations.

Math intuition

Momentum

$$W_{t+1} = W_t - V_t$$

where

$$V_t = \beta V_{t-1} + \alpha \nabla W_t$$

In momentum, next position depends on 2 factors (simultaneous process)
1. History velocity (βV_{t-1})
2. Gradient at that point ($\alpha \nabla W_t$)

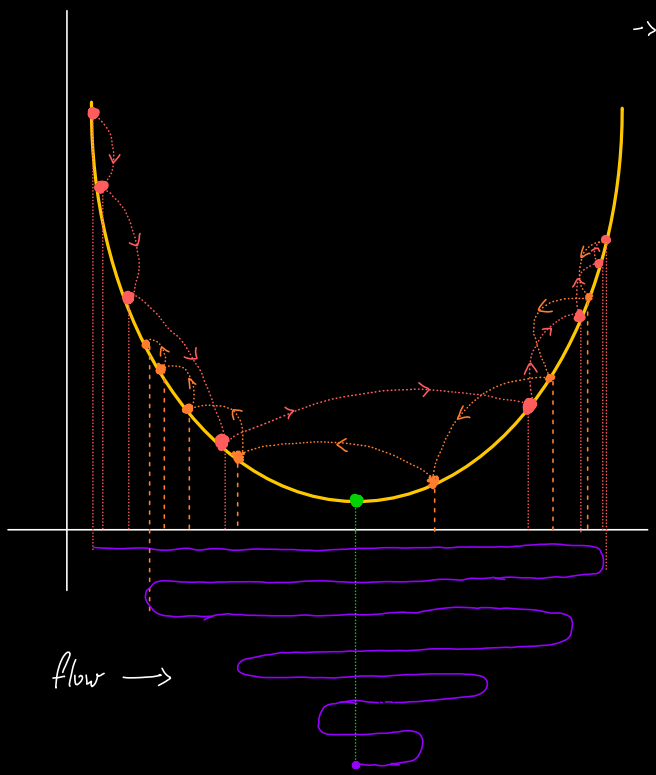
In NAG, first we will update our position based on momentum (look ahead term)
-> Based on new position, we will calculate gradient from that location.

$$W_{la} = W_t - \beta V_{t-1}$$

where W_{la} = look-ahead

$$V_t = \beta V_{t-1} + \alpha \nabla W_{la}$$

$$W_{t+1} = W_t - V_t$$



In simple words...

Step : 1 follow the momentum & go to that direction.

Step : 2 after reaching at the location calculate the gradient at new location.

Disadvantages

-> local minima trap.

```
sgd_optimizer = tf.keras.optimizers.SGD(
    learning_rate=lr_schedule,
    momentum=0.9,
    nesterov=True,
    clipnorm=1.0  # Gradient clipping to avoid exploding gradients
)
```

SGD	SGD + momentum	NAG
momentum = 0.0	momentum = 0.9 (decay factor)	momentum = 0.9
nesterov = False	nesterov = False	Nesterov = True

AdaGrad (adaptive gradient)

→ learning-rate will keep changing based on situation.

Usefull scenarios

→ Scale of i/p features are different. eg. (age, salary)
→ Sparse data (Most values are 0)

→ Problem with Sparse data.

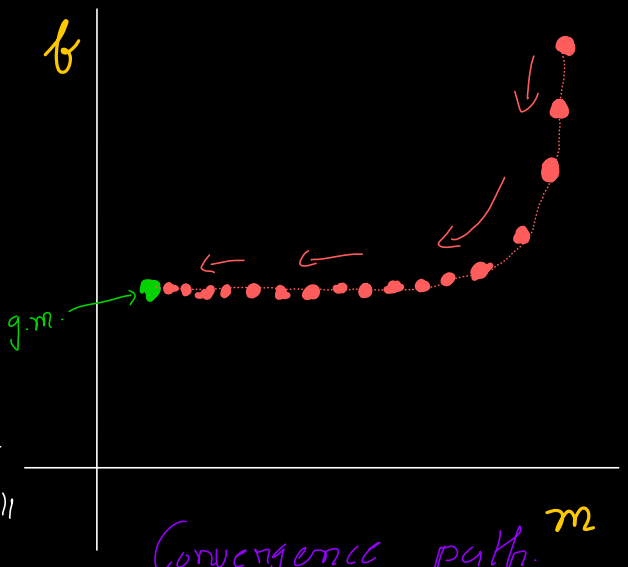
- **Elongated bowl Problem.**
  → Slope changes only on one axis but keeps constant in another axis.

Why it happens..?

→ Suppose we have a dataset like x, y where x is sparse column by making prediction $mx + b$, the term mx will become zero in majority cases & b will dominate in some data-points. hence weight (w) will converge slowly.

$$w = w - \alpha \frac{\partial L}{\partial w}$$ (slow convergence)

$$b = b - \alpha \frac{\partial L}{\partial b}$$ (fast convergence)



→ As graph shows, initially b will converge very fast, whereas due to sparse values, convergence in m (horizontal) direction becomes too slow untill b reaches it's optimal location.

→ hence majority of values in x are zero, the convergence in m (horizontal direction) will be very slow.

→ This is the problem, we don't want such behaviour, we want that convergence in both direction must be **equal**.

→ In this situation we have one last option **learning_rate**, so in AdaGrad there will be **different learning rate for individual parameters.**

→ In this situation, to balance convergence of each parameter (w, b) we will use different learning rate for individual one,

Ex:- $w = w - \alpha \frac{\partial L}{\partial w}$ (slow convergence) → Set high learning rate

$b = b - \alpha \frac{\partial L}{\partial b}$ (fast convergence) → Set low learning rate

Math formula

$$W_{t+1} = W_t - \frac{\alpha}{\sqrt{V_t + \epsilon}} \nabla W_t \longrightarrow \frac{\partial L}{\partial w}$$

Same for b also...

ϵ → small number

$$V_t = V_{t-1} + (\nabla W_t)^2$$

(V_t → squared sum of past gradients)

Disadvantages

→ Not useful in NN. (Can be useful in Linear Regression but not for complex problems)

→ It can reach near to solution but never to exact solution. due to decreasing learning rate.

Solution
- RMSProp
- Adam

RMSProp (Root Mean Squared Propagation)

→ Improved version of AdaGrad.

→ AdaGrad reduces learning-rate continously but at some point learning-rate becomes too small & due to this reason, convergence can't happen, due to this problem RMSProp is here to solve the problem.

Adagrad

$$W_{t+1} = W_t - \frac{\alpha}{\sqrt{V_t + \epsilon}} \nabla W_t$$

$$V_t = V_{t-1} + (\nabla W_t)^2$$

for b

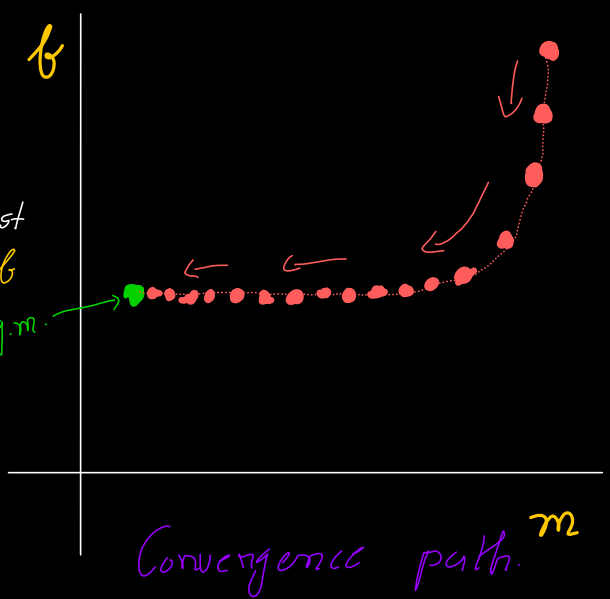
$$b_{t+1} = b_t - \frac{\alpha}{\sqrt{V_t + \epsilon}} \nabla b_t$$

$$V_t = V_{t-1} + (\nabla b_t)^2$$

→ In beginning within few epochs convergence of b will be accelerated & there will be almost no convergence in m.

→ But after some times due to too many past convergence of b, eventuly learning rate of b will become too smaller. hence no convergence will occur in b.

→ So we need to control the explotion of V_t. that's what RMSProp does.



RMSProp

$$W_{t+1} = W_t - \frac{\alpha}{\sqrt{V_t + \epsilon}} \nabla W_t$$

$\beta = 0.95$ (Most cases)

$$V_t = \beta V_{t-1} + (1-\beta)(\nabla W_t)^2$$

→ Exponentially decaying average

→ Giving more importance to recent gradient instead of giving more importance to olden gradients also.

→ This mechanism won't reduce learning-rate gradually after so... epochs.

→ hence V_t never shoot.

$V_0 = 0$

$V_1 = 0.95 \times 0 + 0.05(\nabla W_1)^2$

$V_2 = 0.95 \times 0.05(\nabla W_1)^2 + 0.05(\nabla W_2)^2$

$V_3 = \underbrace{0.95 \times 0.95 \times 0.05(\nabla W_1)^2}_{\text{epoch-1}} + \underbrace{0.95 \times 0.05(\nabla W_2)^2}_{\text{epoch-2}} + \underbrace{0.05(\nabla W_3)^2}_{\text{epoch-3}}$

→ This is one of the best optimization technique with no major disadvantages.

→ Widely used optimizer before Adam.

Adam (Adaptive Moment Estimation) Optimizer

-> Top performance — ANN, CNN, RNN

-> Till now whichever optimizers we have seen, they have used 2 main techniques 1. Momentum, 2. learning rate decay but Adam uses both of them.

Math intuition

$$W_{t+1} = W_t - \frac{\alpha}{\sqrt{V_t + \epsilon}} \times m_t$$

where

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \nabla W_t \quad \} \text{ momentum}$$

$$V_t = \beta_2 V_{t-1} + (1-\beta_2)(\nabla W_t)^2 \quad \} \text{ Adagrad}$$

Bias correction

$\beta_1 = 0.9$, $\beta_2 = 0.99$ } By default in keras

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{V}_t = \frac{V_t}{1-\beta_2^t}$$

Where t = number of epochs