07-Oct-2024

(1.) Momentum:

gradient descent with momentum:

- the problem with all the variant of gradient descent is that it takes lot of time to pass through the gentle slope. This is because at gentle slope gradient is very small so update become slow.
- To solve this problem we will use the idea of momentum incorporated to gradient descent.

momentum:

Momentum is like a ball rolling down hill. The ball will gain momentum as it rolls down the hill.

- in simple term, we want to reach to the destination which entirely new for us. what will we do. we will aske direction from the person nearby.
- That person will direct us in some direction, so we move in that direction slowly. After reaching certain distance again we ask the another person for the direction and that person also point us in the some direction; then we will move in that direction with more acceleration.
- So our acceleration will definitely increase. as we gain information for the later person and earlier person are same. This phenomenon is called momentum.

Why use the momentum optimization?

if the our graph is non convex then these problem is resolve by momentum.

- high curvature
- consistent gradient
- Noisy gradient (local minima)

Momentum Optimization (Mathematical background):

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

weight update in GD

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

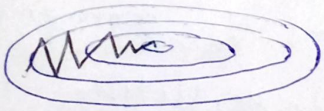
$V \rightarrow$ velocity

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

weight update in GD with momentum

$$0 < \beta < 1$$

here history of past velocity use as the momentum



SGD without momentum

here updates takes longer vertical steps [slower learning] than horizontal step [faster learning]

SGD with momentum

here updates takes longer horizontal steps [faster learning] than vertical step [slower learning].

if $\beta = 0$ then it work as a SGD

if $\beta = 1$ then there is no decay, the point is oscillate

- if $\beta \geq 0.9$ then $\frac{1}{1-\beta} = 10$

current velocity is equal to average of past 10 velocity.

Problems with Momentum:

- Momentum based gradient descent oscillates in and out of the mining valley as the momentum carries it out of the valley takes a lot of u-turns before finally converging despite these u-turns it ~~will~~ still converges faster than vanilla gradient descent.
- we think of the problem like rolling a ball from the hill top it will rest in the valley for sure but it will oscillate to and fro near the valley before resting.

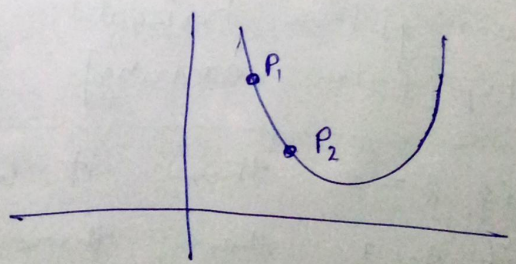
(2.) Nesterov Accelerated Gradient (NAG)

- it is the updraded version of momentum
- It is help to reduce the oscillation in point with the same decay factor that we choose in the momentum

in momentum

$$w_{t+1} = w_t - (\beta V_t + \eta \nabla w_t)$$

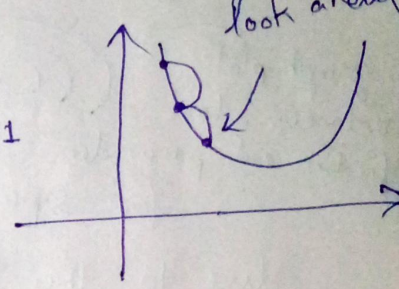
$$w_{t+1} = w_t - \eta \nabla w_t$$



- in SGD without momentum ~~does~~ jump from P_1 to P_2 depends on only. $\eta \Delta W_t$ (gradient)

- But in SGD with momentum jump from P_1 to P_2 depends on two factor $\eta \nabla W_t$ (gradient) and βV_t (velocity) also history

- In NAG first we consider the history of velocity (βV_t) and then we consider the $\eta \nabla W_t$ (gradient) according to that we move forward.



$$W(\text{look ahead}) = W_t - \beta V_{t-1}$$

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

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

$$\text{or} \quad V_t = (W_t - W(\text{look ahead})) + \eta \nabla W_t$$

disadvantages:

- NAG reduces the oscillation due to that may be possibility that we can trapped in local minima.

keras Code:

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False,
    name="SGD", **kwargs)
```

(3.) AdaGrad (Adaptive Gradient)

- it this optimizer the learning rate not fixed
- it is perfectly work with convex but not with non-convex

where we use adagrad?

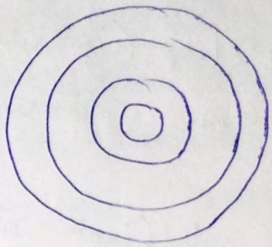
→ if input features scales are different.

→ if feature are sparse (most of the values are zero).

(due to sparse data we can face a elongated ~~bowl~~ bowl problem)

graph plot between loss function and parameter

for sparse data



for non-sparse data.

due to this type of graph the SGD, BGD, momentum, and NAG are not work perfectly

- in gradient descent most of the gradient ~~become~~ zero and the update are very small due to sparse data.

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

$$b = b - \eta \frac{\partial L}{\partial b}$$

→ in adagrad for different parameter we set the different learning rate

if one parameter update is big then their learning rate set as small vice-versa.

here the parameter is w and b

$$W_{t+1} = W_t - \frac{\eta \nabla W_t}{\sqrt{V_t + \varepsilon}}$$

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

some formula for 'b' also

Disadvantages:

- adagrad reaches the nearby the optimal solution but not ~~reaches the~~ converge at minima due to we decreases the learning rate

(4.) RMSProp (Root mean square propagation):

- it is improved version of Adagrad
- in RMSprop

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

$$W_{t+1} = W_t - \frac{\eta \nabla W_t}{\sqrt{V_t + \varepsilon}}$$

example

$V_0 = 0$

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

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

$V_3 = \underbrace{0.95 \times 0.95 \times 0.05 (\nabla W_1)^2}_{\text{Smallest}} + 0.95 \times 0.05 (\nabla W_2)^2 + \underbrace{0.05 (\nabla W_3)^2}_{\text{largest}}$

- it is easily to converge on global minima with convex optimization as well as non-convex optimization.

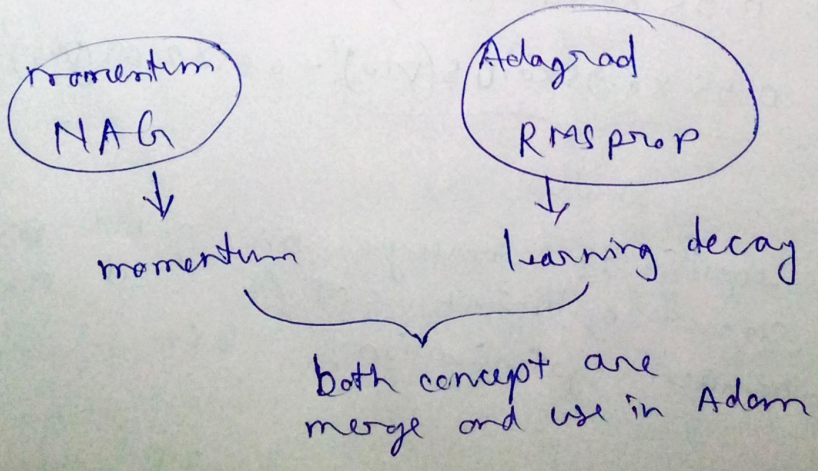
Disadvantages:

There is no disadvantages of this optimizer.

- AdaDelta and RMSprop both are small enhancement over AdaGrad.
- problem of slower convergence is addressed by these algorithms.
- idea of AdaDelta and RMSprop is that instead simply using the square root summation of squared history gradient why can't we use exponential decaying average (eda)/ exponential moving average/running average.
- So by applying the running average we are controlling the growth of the denominator
- In simpler words, we are using the mean of the history gradient to decay the learning rate

(5.) Adam (Adaptive Moment Estimation):

- currently most powerful optimization technique



Mathematical Formulation:

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{V_t + \varepsilon}} * m_t$$

where

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

$V_t = \beta_2 V_{t-1} + (1-\beta_2)(\nabla W_t)^2$ ⟶ adagrad.

Bias correction:

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

Here T ⟶ epoch number.

usually, $\beta_1 = 0.9$ $\beta_2 = 0.99$

Optimizer	when to use it
(i) Mini Batch SGD	use when network is small and shallo
(ii) momentum (SGD)	works well in most of the cases but slightly slower in convergence due to oscillation.
(iii) Adagrad Adadelta and RMS prop	use when there is sparse data.
(iv) Adam and its variation	Always recommended