Deep learning Day-4

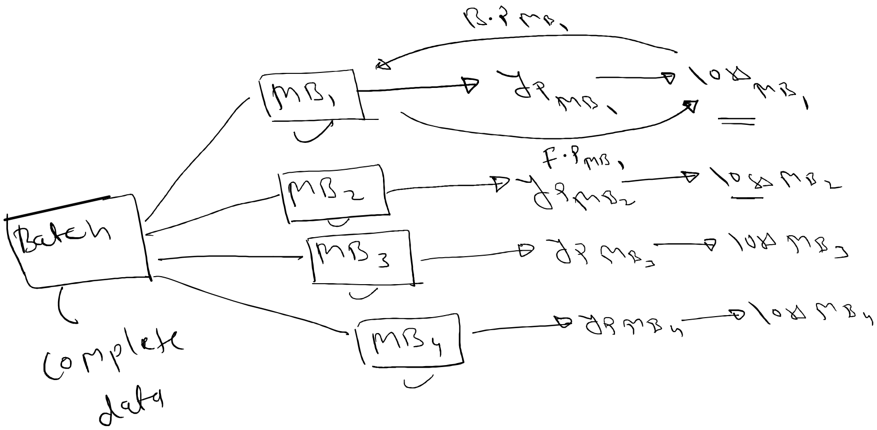
(3) Mini Batch Stochastic gradient Decent algorithm

MBSGDA

BGDA + SGDA

BGDA → learning curve smoothness

SGDA → Step by step learning



min [loss $_{MB_1}$, loss $_{MB_2}$, loss $_{MB_3}$, loss $_{MB_4}$]

→ min loss

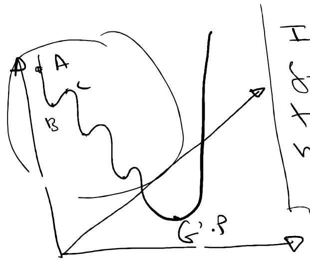
→ Best w, b

→ Best Summation func

→ Activation func

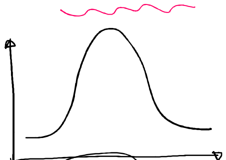
→ final yp

Parallel and distribution learning



In MBSGDA we learn from mini Batch at a time, hence now we have sufficient amount of learning to overcome local minima point and we ensure that we will reach to Global minima.

* Random sampling



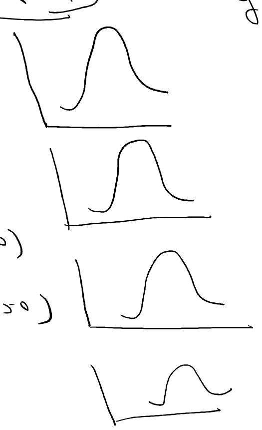
Complete data (1000)

Batch1 (250)

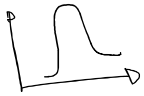
Batch2 (250)

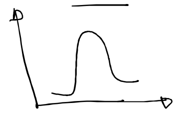
Batch3 (250)

Batch4 (250)



1000 → 10 mini Batches — 1 mini Batch → 100





* Advantage of MBSGDA :-

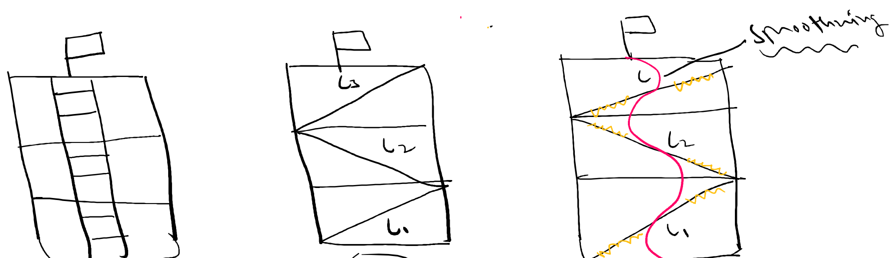
① Compare to BGDA, MBSGDA required less resources and time to achieve convergence.

② Since we are learning from mini Batch at a time, we have sufficient amount of learning to over come local minima hence we ensure that we will reach to Global minima.

* Drawback of MBSGDA :-

① Noise is still present in learning curve.

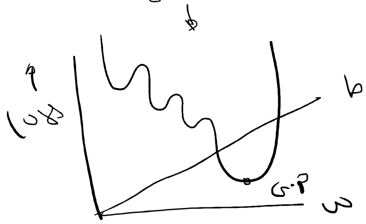
* Why we don't want noise in our learning curve ?

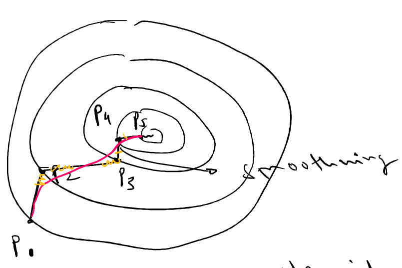


(A) time ↓ effort ↑

(B) time ↑ effort ↓

(C) time ↓ effort ↓





Noise → time ↓ ⌉ time complexity

Noise → time ↓ → time complexity

— effort ↓ → Resources.

(4) Mini Batch stochastic gradient Descent algorithm with momentum.

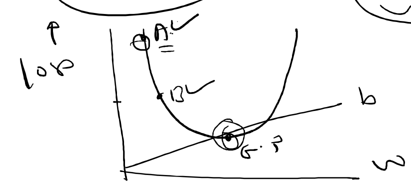
smoothing

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$$

$$b_{new} = b_{old} - \alpha \frac{\partial L}{\partial b}$$

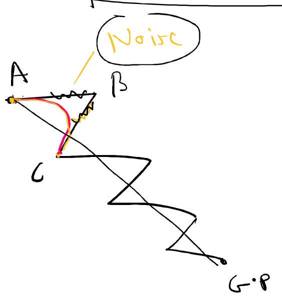
momentum

↑ Gradient → Potential energy ↑



$\frac{\partial L}{\partial w}$ → Changes in loss with respective weight.

learning inertia



$$w_{new} = w_{old} - \alpha \times v_{dw}$$

$$b_{new} = b_{old} - \alpha \times v_{db}$$

v_{dw} or v_{db} → velocity component

$$v_{dw_t} = \beta \times v_{dw_{t-1}} + (1-\beta)\frac{\partial L}{\partial w}$$

$$v_{db_t} = \beta \times v_{db_{t-1}} + (1-\beta)\frac{\partial L}{\partial b}$$

where

t → no of hidden layer

Range → β → smoothing parameter (0 to 1)

→ 0 to 1

v_{dw} or v_{db} → velocity component

Case I → $\beta = 0$ ×

$$v_{dw_t} = \beta \times v_{dw_{t-1}} + (1-\beta)\frac{\partial L}{\partial w}$$

$$= 0 \times v_{dw_{t-1}} + 1 \times \frac{\partial L}{\partial w}$$

$$v_{dw} = \frac{\partial L}{\partial w}$$ → No smoothing

Case II → $\beta = 1$ ✗

$$Vdw_t = \beta \times Vdw_{t-1} + \underbrace{(1-\beta)\frac{\partial L}{\partial w}}_{0}$$

$$\boxed{Vdw_t = Vdw_{t-1}}$$ → Full smoothening

1 2 3

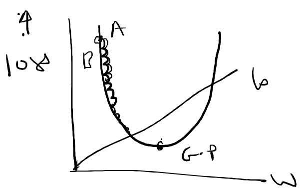
Case III → $\beta = 0.98$

$$Vdw_t = \beta \times Vdw_{t-1} + (1-\beta)\frac{\partial L}{\partial w}$$ — G.P

$$\boxed{Vdw_t = \underbrace{0.98\, Vdw_{t-1}}_{\text{I Magnitude}} + \underbrace{0.02\frac{\partial L}{\partial w}}_{\text{II Direction}}}$$

* Best optimization function for weight and bias → MBSGDA with momentum

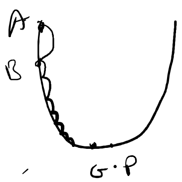
* Why we need to update learning rate dynamically in Neural Network?



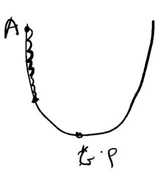
Learning Rate — constant

Learning step

Rate



Case I → If learning rate is too small? (0.001



→ training time increase exponentially.

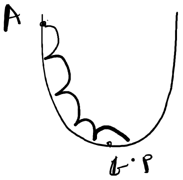
→ We face vanishing gradient issue.

Vanishing Gradient → We never achieve global minima

Case II → If learning rate is too high? (0.1

Case II $\longrightarrow$ If learning rate is too high.

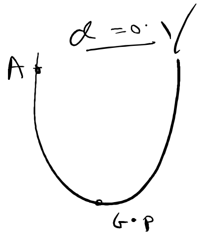
$\hookrightarrow 0.1$



$\rightarrow$ we face over shooting issue

$\rightarrow$ due to over shooting we face exploding gradient issue.

Exploding gradient



$\alpha = 0.1 \longrightarrow$ too high $\downarrow$

$\alpha = 0.01 \longrightarrow$ small $\uparrow$

learning rate

$\hookrightarrow$ dynamically update.

(1) Ada grad (Adaptive gradient)

(2) Ada Delta

$\hookrightarrow$ it is also called as RMS prop (Root mean squared propogation

(1) Ada grad

$$w_{new} = w_{old} - \alpha_{new} \frac{\partial L}{\partial w}$$

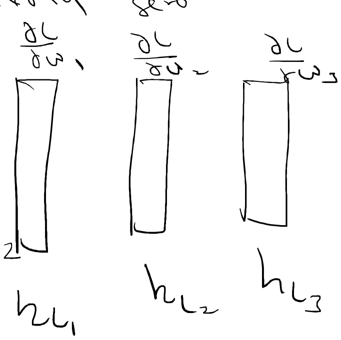
$$b_{new} = b_{old} - \alpha_{new} \frac{\partial L}{\partial b}$$

$$\boxed{\alpha_{new} = \frac{\alpha_{old}}{\sqrt{\eta + \epsilon}}}$$

$\rightarrow$ where $\epsilon \simeq e^{-6}$ (small positive number)

we use ϵ in equation to avoid zero divisional error.

$$\eta = \sum_{i=1}^{n} \left(\frac{\partial L}{\partial w_i}\right)^2$$



$$A \quad \eta = \left(\frac{\partial L}{\partial w_1}\right)^2 + \left(\frac{\partial L}{\partial w_2}\right)^2 + \left(\frac{\partial L}{\partial w_3}\right)^2$$

$\eta \uparrow \longrightarrow \alpha_{new} \downarrow \longrightarrow$ vanishing gradient

* Drawback of Ada grad

→ We can not use Ada grad in case of deep Neural Networks.

(2) Ada Delta (RMS prop)

(Root mean squared propogation)

$\uparrow \eta \longrightarrow \alpha_{new} \downarrow$

$\eta \longrightarrow$ replace $\longrightarrow$ $\underline{S_{dw_t}}$

$$w_{new} = w_{old} - \alpha_{new} \frac{\partial L}{\partial w}$$

$$b_{new} = b_{old} - \alpha_{new} \frac{\partial L}{\partial b}$$

$$\alpha_{new} = \frac{\alpha_{old}}{\sqrt{S_{dw_t} + \epsilon}}$$

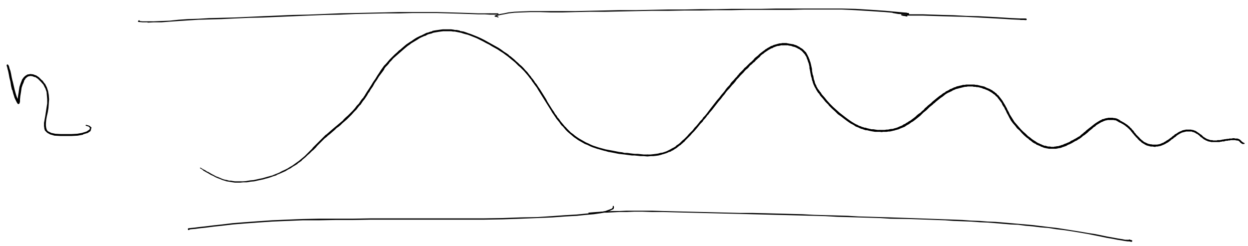
S_{dw_t} = velocity component

$$V_{dw_t} = \beta * V_{dw_{t-1}} + (1-\beta) \frac{\partial L}{\partial w}$$

$$\boxed{S_{dw_t} = \beta * S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)^2}$$

$$S_{dw_t} = \beta * S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)$$

$\beta \rightarrow$ smoothing parameter



→ Best optimization function for learning rate → Ada Delta

(RMS prop)

* Best optimization function for weight and bias ?

↳ MBSGD A with momentum

+

* Best optimization function for learning rate ?

↳ Ada Delta (RMS prop)

= ADAM