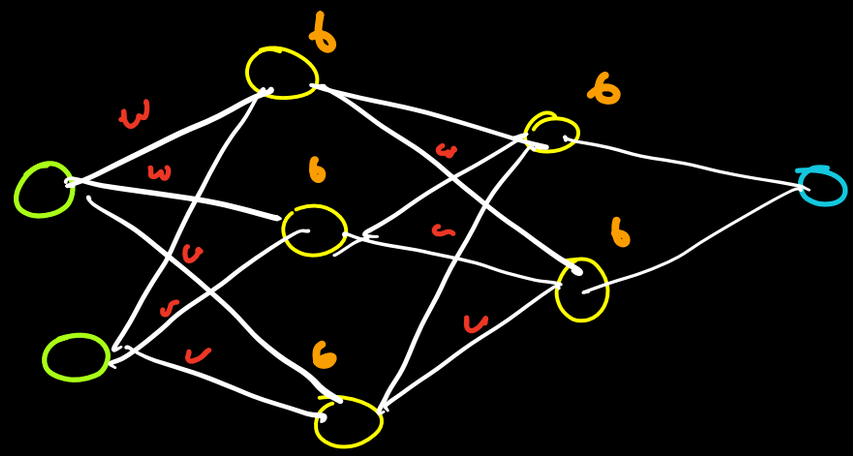
Weight Initialisation & Optimisers

[Neural Network]

Q: Why random init?. Why not constant?



→ 0
→ 1
→ 75 ?

if all weights are same in beginning then, all gradients will be same for every-one. Then all updates will be same. Hence network will never learn.

→ A-symmetry is very important.

Random Init

$$w_{ij}^{k} \sim N(0, \sigma)$$ OR

$$w_{ij}^{k} \sim U(-1, 1)$$

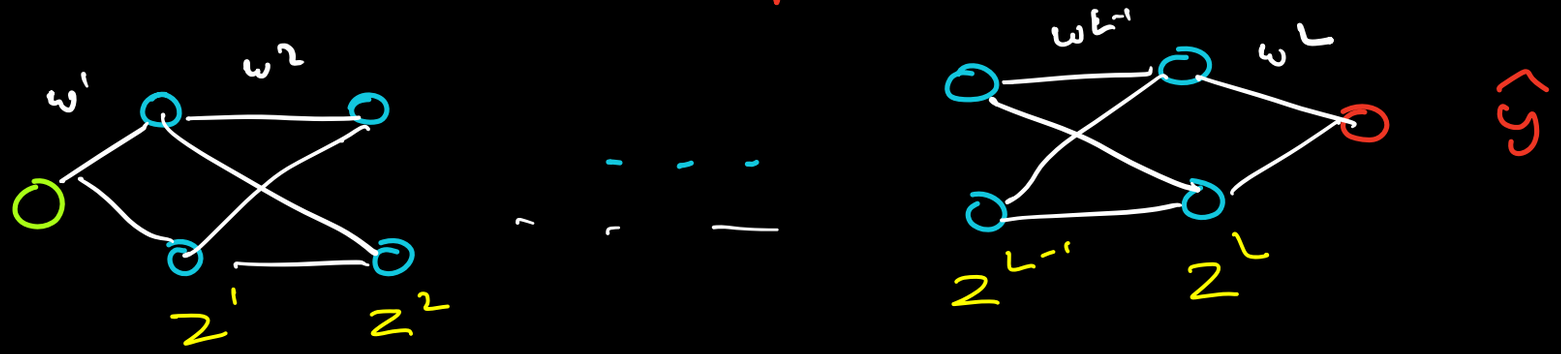
But this causes the problem of exploding gradients in deep NN.

↓

large number of hidden layers.

Vanishing / Exploding Gradients

Let's take a deep NN. (simple one)



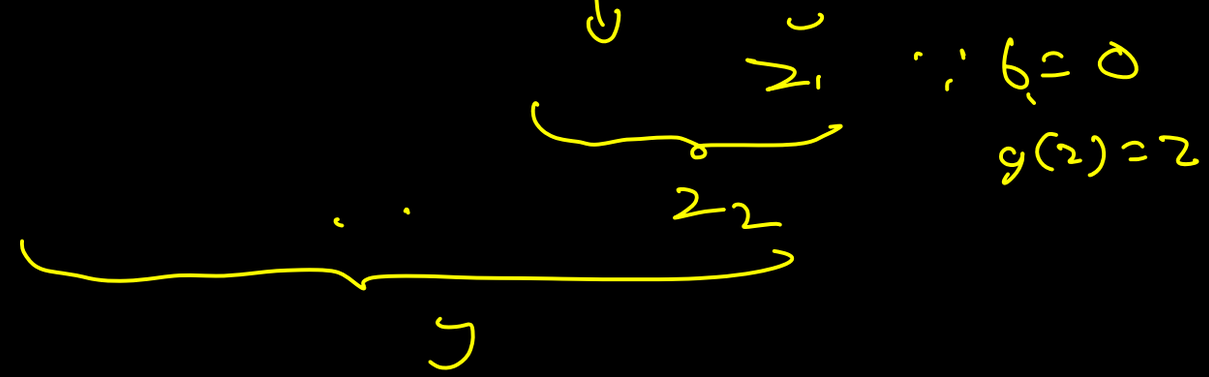
Let's say all $b_i = 0$

and linear activation, so $g(z) = z$

$\therefore \quad z^1 = w^1 x$

$\therefore \quad z^2 = w^2 z^1 \rightarrow w^2 w^1 x$

$\therefore \quad y = w^L \cdot w^{L-1} \cdots w^2 \cdot \underbrace{w^1 x}$



We see too many weight multiplies

So if w_s are > 1, then,

Since L is large

$(1.1)^L \rightarrow \infty$ Exploding

$(0.8)^L \rightarrow 0$ Vanishing

This will also happen is back prop!!

Solution

1. Relu → Since its gradient is '1' on +ve side, it mitigates. $1^n = 1$ ✓
   However since it does not dampen activation, if weights are large it can cause exploding gradient.
2. Smart weight initialisation:
   There is not a strong theoretical reason but the following have been shown to work experimentally

1. He Init

$$\rightarrow \quad N\left(0, \frac{2}{n^{l-1}}\right)$$

↳ # neurons in prev layer

$$\rightarrow \quad U\left[\frac{-\sqrt{6}}{\sqrt{n^{l-1}}}, \frac{\sqrt{6}}{\sqrt{n^{l-1}}}\right]$$

2. Glorot Init.

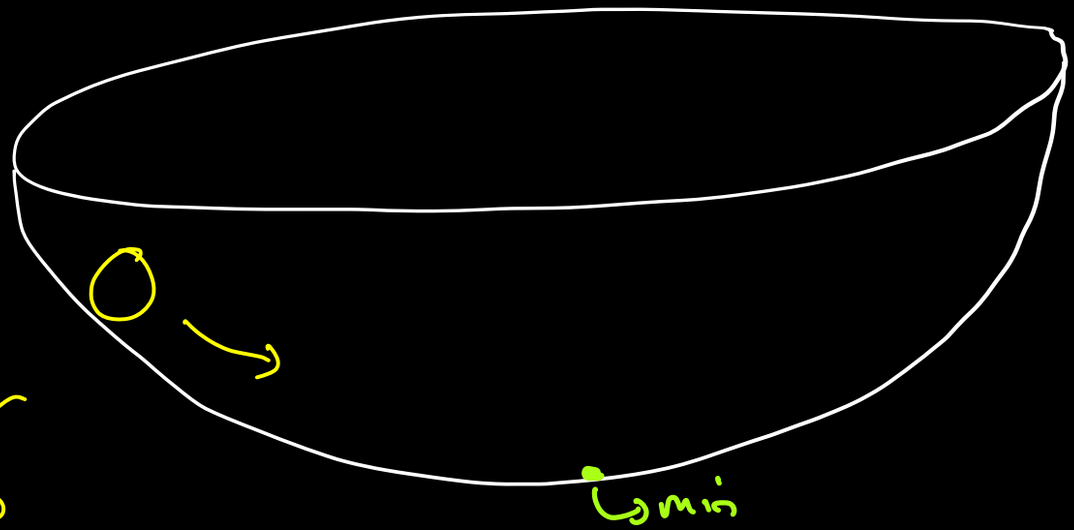
$$\rightarrow \quad N\left(0, \frac{2}{n^{l-1} + n^{l}}\right)$$

$$\rightarrow \quad U\left(\frac{-\sqrt{6}}{\sqrt{n^{l-1} + n^{l}}}, \frac{\sqrt{6}}{\sqrt{n^{l-1} + n^{l}}}\right)$$

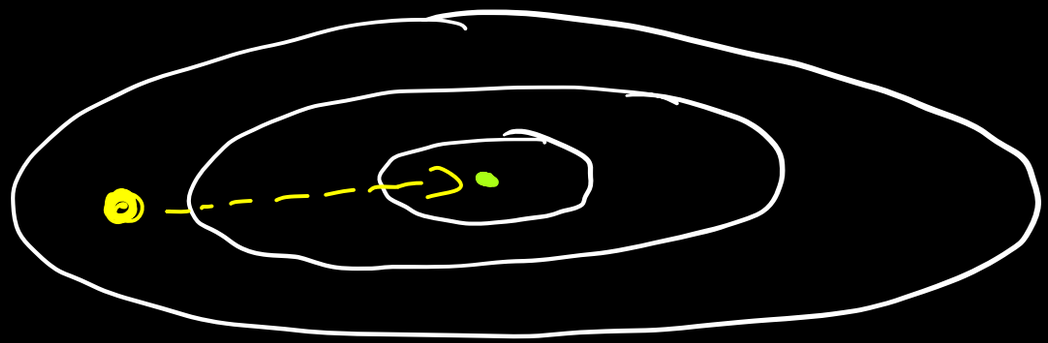
→ code:

Optimizers

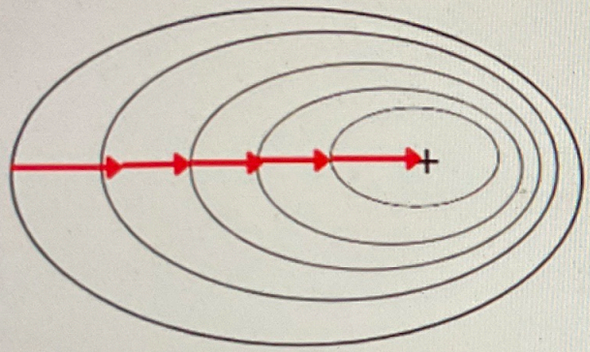
→ imagine a bowl



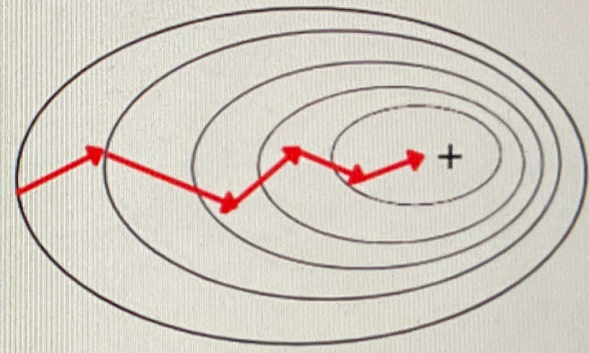
Contour plot (shadow)



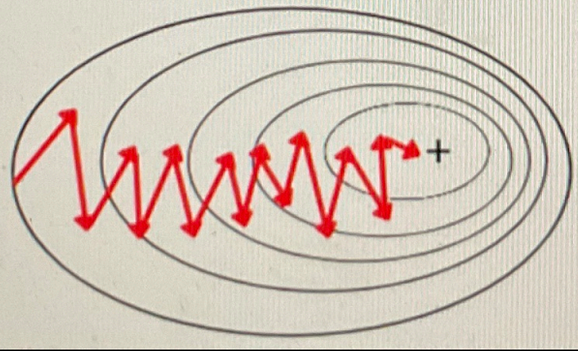
Batch Gradient Descent

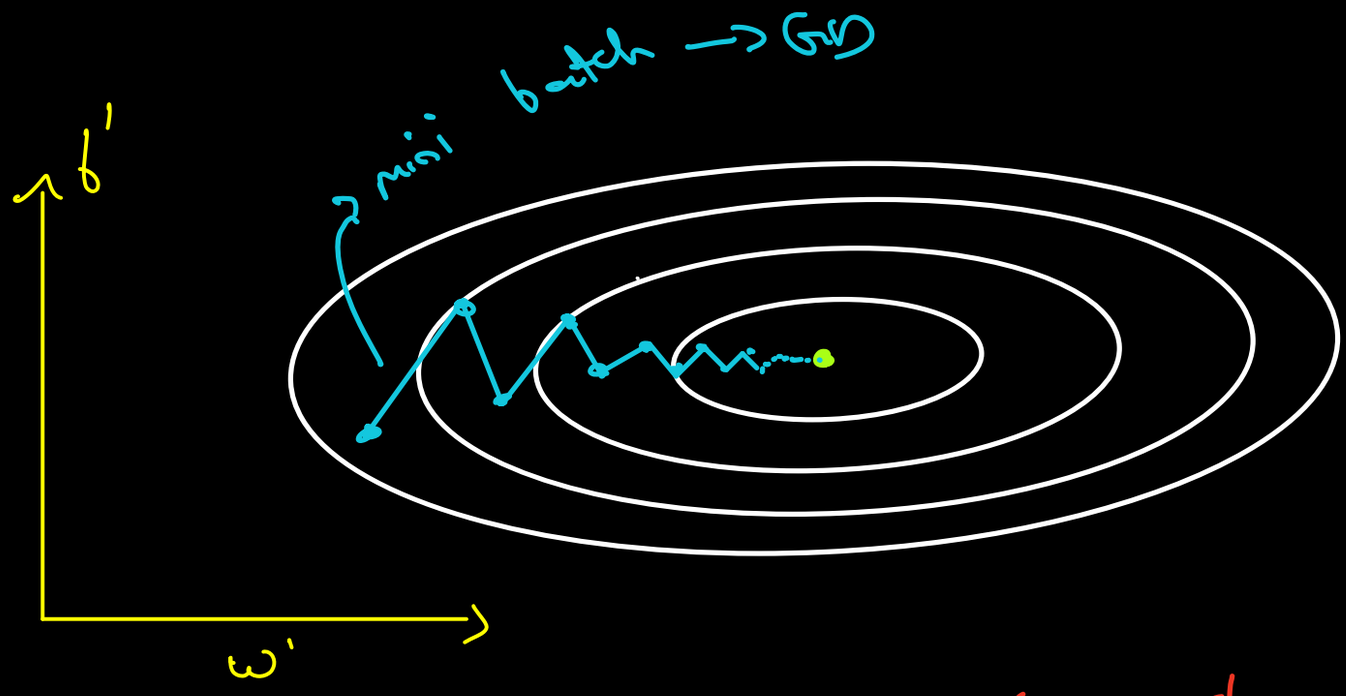


Mini-Batch Gradient Descent



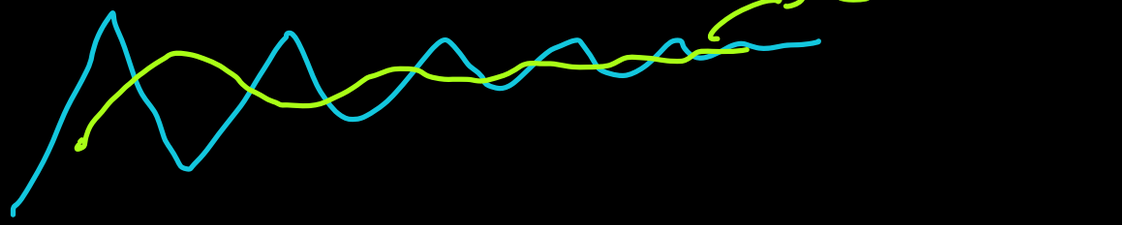
Stochastic Gradient Descent





→ Too many oscilations, how to reduce these?

Simple Exponential Smoothing



A.K.A
Exponential
moving average

How does this work?

y_1 y_2 y_3 y_4 y_5 ← actual value of signal

smooth → y_1 ↓

$\alpha y_2 + (1-\alpha) y_1$ ↓

$\alpha y_3 + (1-\alpha) y_2 + \alpha(1-\alpha) y_1 \ldots$

- α: more weight
- $(1-\alpha)$: less weight
- $\alpha(1-\alpha)$: lesser weight

Current value is influenced by past values!

Where I am now + where I have been before ?

Recursive Eq^

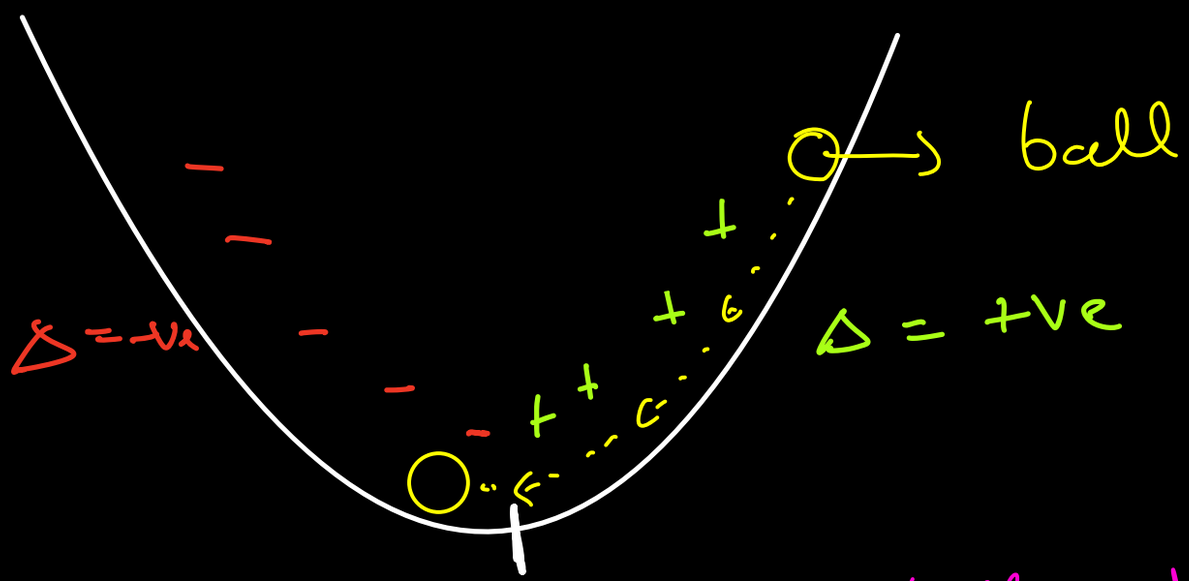
$$\hat{y}_t = \beta y_t + (1-\beta)\hat{y}_{t-1}$$

$$= \beta y_t + (1-\beta)\left(y_{t-1} + \beta(\hat{y}_{t-2})\right)$$

$$= \beta y_t + (1-\beta)(y_{t-1}) + \beta(1-\beta)(\hat{y}_{t-2})$$

$$\beta(1-\beta)y_{t-2} + \beta(1-\beta)^2\hat{y}_{t-3}$$

Momentum based GD



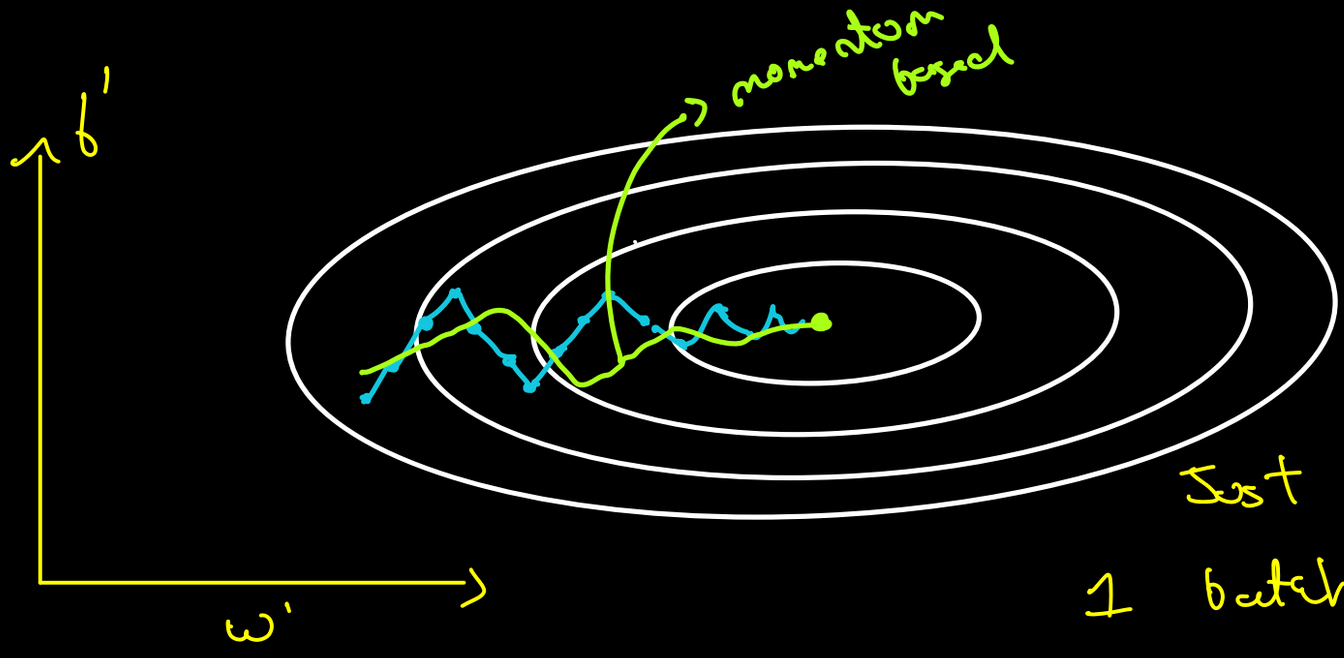
Q. Does the ball change direction immediately ?! (Physics)

→ No, due to momentum

It was going in the same direction for

many steps. Q: Should it change direction suddenly

think in terms of mini-batch GD



No sudden changes

↓

Just because 1 batch said to go in diff direction, we will not go!

How to accomplish this?

Normal GD:

$$w^{t+1} = w^t - \alpha \frac{\partial L}{\partial w}$$

$$= w^t - \alpha \, dw$$

$$b^{t+1} = b^t - \alpha \, db$$

Momentum based GD

$$V_{dw^k} = \beta \underbrace{V_{dw^{k-1}}}_{\text{velocity}} + (1-\beta) \overbrace{dw^k}^{\text{acceleratn}}$$

$$V_{db^k} = \beta V_{db^{k-1}} + (1-\beta) db^k$$

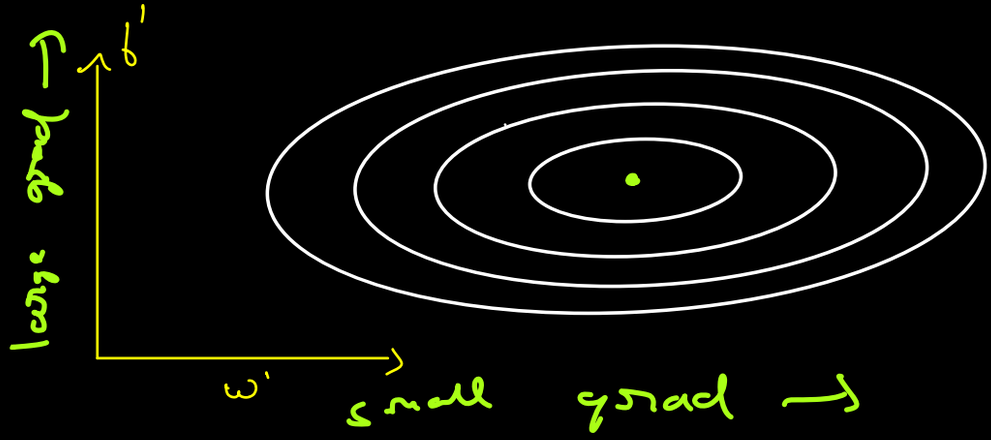
$$w^{k+1} = w^k - \alpha V_{dw^k}$$

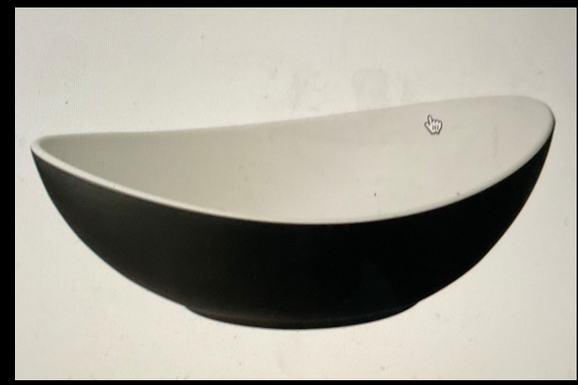
$$b^{k+1} = b^k - \alpha V_{db^k}$$

-> code

RMS prop

there is still one more issue.





$db \gg dw$

But learning rate is same !

We want to take smaller steps in b direction

$$S_{dw^k} = \beta S_{dw^{k-1}} + (1-\beta)(dw^k)^2$$

$$S_{db^k} = \beta S_{db^{k-1}} + (1-\beta)(db^k)^2$$

still,

$$S_{db} \gg\gg S_{dw}$$

Update →

$$w^{k+1} = w^k - \alpha \frac{dw^k}{\sqrt{S_{dw^k} + \varepsilon}}$$

→ To avoid div by 0

$$b^{k+1} = b^k - \alpha \frac{db^k}{\sqrt{S_{db^k} + \varepsilon}}$$

Now, Step size will be more limited!!

db is divided by a large number, so it becomes smaller

Adaptive Momentum Estimation (ADAM)

Combination of both !

$$V_{dw^k} = \beta_1 V_{dw^{k-1}} + (1-\beta_1)\, dw^k$$

$$V_{db^k} = \beta_1 V_{db^{k-1}} + (1-\beta_1)\, db^k$$

$$S_{dw^k} = \beta_2 S_{dw^{k-1}} + (1-\beta_2)\,(dw^k)^2$$

$$S_{db^k} = \beta_2 S_{db^{k-1}} + (1-\beta_2)\,(db^k)^2$$

Update :

$$w^{k+1} = w^k - \alpha \frac{V_{dw^k}}{\sqrt{S_{dw^k}} + \varepsilon}$$

$$b^{k+1} = b^k - \alpha \frac{V_{db^k}}{\sqrt{S_{db^k}} + \varepsilon}$$

This is most popular and tensorflow default.

Recommended values:

$\alpha \rightarrow$ tune

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

↳ no need to change.

Final note:

Since we are using exponential avg,

$$V^{k+1} = \beta V^{k} + (1-\beta) dw$$

↓

what is $V^{0} = ?$

→ 0

because of init with 0, for the first few steps, there is under estimation of momentum, hence there is a small modification → Refer Post Recap!