dec1: Mini batch Gradient descent:

$$X = \underset{(n_x, m)}{[x^{(1)}\ x^{(2)}\ \cdots\ x^{(m)}]} \quad Y = \underset{(1, m)}{[y^{(1)}\ y^{(2)}\ \cdots\ y^{(m)}]}$$

if $m = 5{,}000{,}000$, the G.D is very slow.

sol. mini batches of 1000 each

$$X = \Big[\underbrace{x^{(1)}\ x^{(2)}\ \cdots\ x^{(1000)}}_{X^{\{1\}}\ (n_x, 1000)} \Big| \underbrace{x^{(1001)}\ \cdots\ x^{(2000)}}_{X^{\{2\}}\ (n_x, 1000)} \Big| \cdots \Big| \underbrace{\cdots\ x^{(m)}}_{X^{\{5000\}}\ (n_x, 1000)} \Big]$$

$$Y = \Big[\underbrace{y^{(1)}\ y^{(2)}\ \cdots\ y^{(1000)}}_{Y^{\{1\}}\ (1000, 1)} \Big| \underbrace{y^{(1001)}\ \cdots\ y^{(2000)}}_{Y^{\{2\}}\ (1000, 1)} \Big| \cdots \Big| \underbrace{y^{(m)}}_{Y^{\{5000\}}\ (1000, 1)} \Big]$$

mini batch t : $\underset{(n_x, 1000)}{X^{\{t\}}}$, $\underset{(1000, 1)}{Y^{\{t\}}}$

Algorithm:- (Mini Batch GD)

for $t = 1, \ldots, 5000$:

{

Forward prop on $X^{\{t\}}$

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$\vdots$$

$$A^{[L]} = g^{[L]}(Z^{[L]})$$

(vectorized implementation (1000 examples))

Compute cost function

$$J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2(m=1000)} \sum_{l} \|w^{[l]}\|_F^2$$

($X^{\{t\}}$, $Y^{\{t\}}$)

Backprop to compute gradient w.r.to $J^{\{t\}}$ (using $X^{\{t\}}, Y^{\{t\}}$)

$$w^{[l]} = w^{[l]} - \alpha dw^{[l]}$$

$$b^{[l]} = b^{[l]} - \alpha db^{[l]}$$

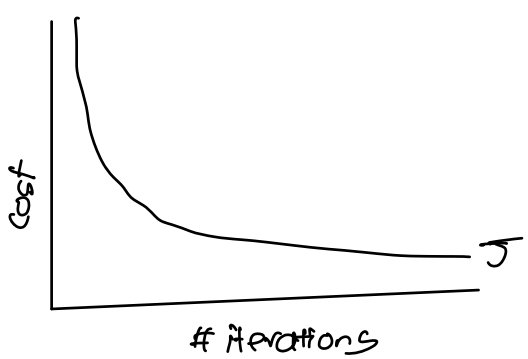
}

1 epoch = single pass through training set

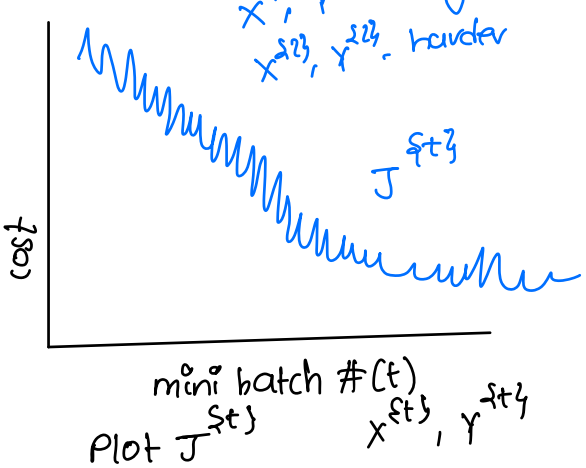
implement 1 step of GD using $X^{\{t\}}, Y^{\{t\}}$

Deca: Understanding mini batch GD:-

Batch GD



Mini batch GD



Plot $J^{\{t\}}$ $X^{\{t\}}, Y^{\{t\}}$

choose mini batch size

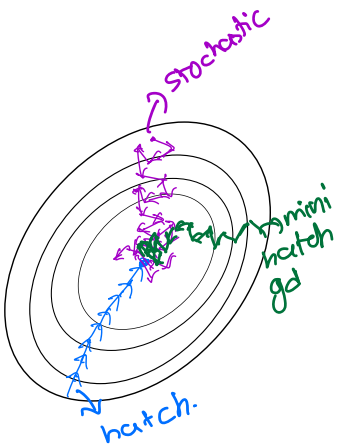
if mini batch size = m ⇒ batch gradient descent.

$(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

if mini batch size = 1 ⇒ stochastic gradient descent

$(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$

$(X^{\{2\}}, Y^{\{2\}}) = (x^{(2)}, y^{(2)})$



in pratice mini batch size in between 1 and m

batch gd → too long per iteration

stochastic gd → loose speed from vectorization.

mini batch gd → fast learning, vectorization (1000)

→ Make progress without needing to wait to iterate enter training set

each epoch will update 5000 times w

if you have small training set → use batch gradient descent ($m \leq 2000$)

typical mini batch sizes:

64, 128, 256, 512, 1024

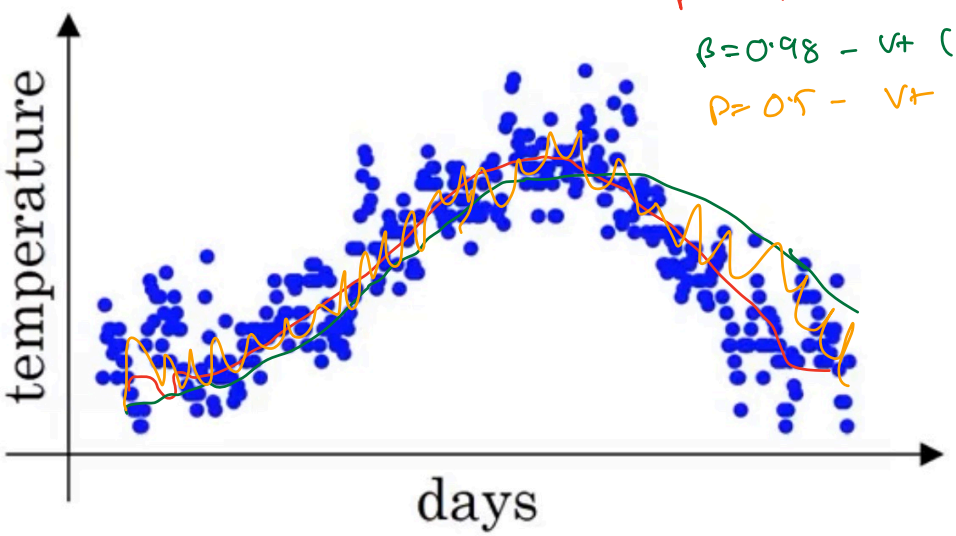
2^6, 2^7, 2^8, 2^9, 2^{10}

Make sure mini batch fits in cpu / GPU memory $(X^{\{t\}}, Y^{\{t\}})$

dec3. Exponentially Weighted Averages:-

temperature in London

$\theta_1 = 40^\circ F \quad \theta_2 = 40^\circ F \quad \theta_3 = 45^\circ F \ldots \theta_{180} = 60^\circ F \ldots$



$\beta = 0.9$ — V_t · moving avg (10 day)

$\beta = 0.98$ — V_t (50 day avg)

$\beta = 0.5$ — V_t (2 day avg)

Moving average (exponentially moving average)

$V_0 = 0$

$V_1 = 0.9 V_0 + 0.1 \theta_1$

$V_2 = 0.9 V_1 + 0.1 \theta_2$

$V_3 = 0.9 V_2 + 0.1 \theta_3$

$\vdots$

$V_t = 0.9 V_{t-1} + 0.1 \theta_t$

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$\beta : 0.9 \approx$ 10 day avg temp

V_t = approximately averaging over $\approx \frac{1}{1-\beta}$ days temperature

Lec4: understanding moving avg

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t \qquad \beta = 0.9$$

$$V_{100} = 0.1\,\theta_{100} + 0.9\,V_{99}$$

$$V_{99} = 0.1\,\theta_{99} + 0.9\,V_{98}$$

$$V_{100} = 0.1\,\theta_{100} + 0.9\,(0.1\,\theta_{99} + 0.9\,(0.1\,\theta_{98} + 0.9\,V_{97}))$$

$$= 0.1\,\theta_{100} + 0.1\times 0.9\,\theta_{99} + 0.1\times(0.9)^2\,\theta_{98} + 0.1\times(0.9)^3\,\theta_{97} + 0.1\times(0.9)^4\,\theta_{96} + \cdots$$

$$0.9^{10} \simeq 0.35 \simeq \frac{1}{e} \quad (10 = \text{days avg}).$$

$$(0.9)^{10} = (1-\varepsilon)^{1/\varepsilon} = 1/e$$

if $\beta = 0.98$ $\quad (0.98)^{50} = 1/e$ (50 day avg).

Implementation

$v_0 = 0$

$v_1 = \beta v_0 + (1-\beta)\theta_1$

$v_2 = \beta v_1 + (1-\beta)\theta_2$

⋮

$v_\theta := 0$

$v_\theta := \beta v_\theta + (1-\beta)\theta_1$

$v_\theta := \beta v_\theta + (1-\beta)\theta_2$

$v_\theta = 0$

Repeat {

$v_\theta := \beta v_\theta + (1-\beta)\theta_t$

}

Lecs: Bias correction in exponential Avg:

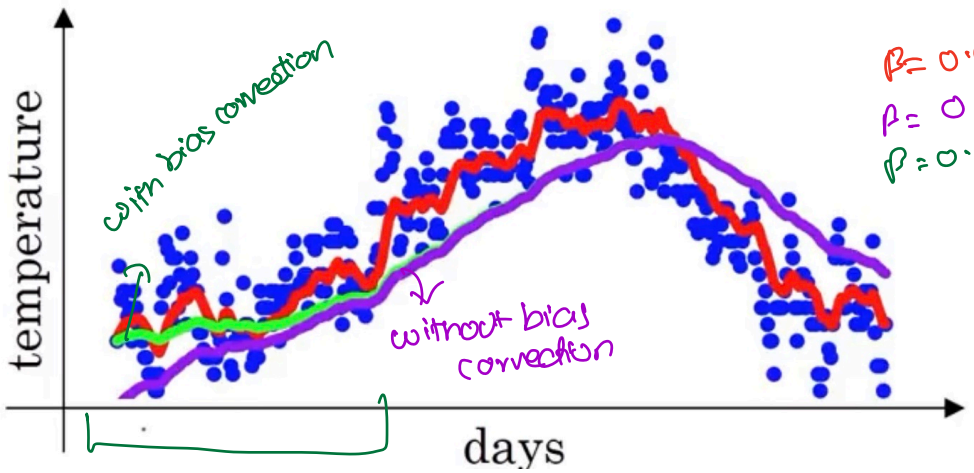
$\theta_1 = 40$

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$V_0 = 0$

$V_1 = 0.98 V_0 + 0.02\theta_1 = 0.02\theta_1 = 0.8$

$V_2 = 0.98 V_1 + 0.02\theta_2 = (0.98)0.8 + 0.02\theta_2$ very less

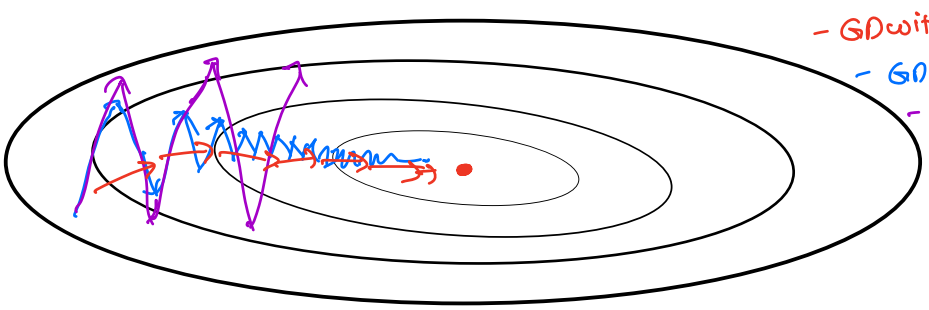


$\frac{V_t}{1-\beta^t}$ t > large ∴ V_t

$t = 2$ $\quad 1-\beta^t = 1-(0.98)^2 = 0.0396$

$$\frac{V_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$ = removes the bais.

Lec 6: Gradient Descent with Momentum :-



vertical axis - slower learning

horizontal axis - faster learning

Momentum:

on iteration t

compute dw, db on current mini batch.

$$v_\theta = \beta v_\theta + (1-\beta)\theta_t$$

$$v_{dw} = \beta v_{dw} + (1-\beta)\,dw$$

$$v_{db} = \beta v_{db} + (1-\beta)\,db \quad \rightarrow \text{acceleration}$$

β ↓ friction, v ↑ velocity

$$w = w - \alpha v_{dw}$$

$$b = b - \alpha v_{db}$$

} smooth steps of gd

Implementation:

on iteration t $\quad v_{dw} = 0$ (dw shape) $\quad v_{db} = 0$ (db shape)

compute dw, db on current mini batch

$$v_{dw} = \beta v_{dw} + (1-\beta)\,dw$$

$$v_{db} = \beta v_{db} + (1-\beta)\,db$$

$$w = w - \alpha v_{dw}$$

$$b = b - \alpha v_{db}$$

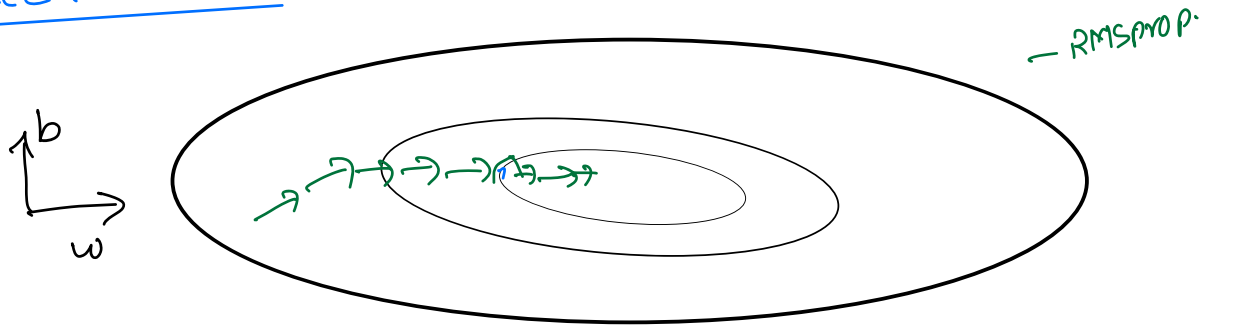
Hyper parameter

α, β

$\beta = 0.9$ (avg over 10 iter)

bias correction $= \dfrac{v_{dw}}{(1-\beta^t)}$ can be included.

dec 7: RMSprop :- (root mean square prop)



on iteration t:

compute dw, db on current mini batch.

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

(small) (large)

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

$$w = w - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon} \quad \text{(large)}$$

$$b = b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon} \quad \text{(reduced)}$$

$\varepsilon = 10^{-8}$

numerical stability :

Lec8. Adam optimization Algorithm:-

Adam = momentum + RMSprop

$V_{dw} = 0 \quad S_{dw} = 0 \quad V_{db} = 0 \quad S_{db} = 0$

on iteration t:

compute dw, db using current mini batch

momentum β_1

RMSprop β_2

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw \qquad V_{db} = \beta_1 V_{db} + (1-\beta_1) db$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2 \qquad S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$$

$$V_{dw}^{corrected} = \frac{V_{dw}}{(1-\beta_1^t)} \qquad V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_1^t)}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{(1-\beta_2^t)} \qquad S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$$

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

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

Hyperparameter:

α: needs to be tune

β_1: 0.9 (momentum) $(dw)\ (db)$ (1st momentum)

β_2: 0.999 (RMSprop) $(dw^2)\ (db^2)$ (2nd momentum)

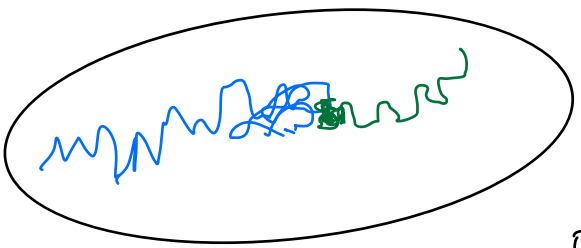
ε: 10^{-8}

Adam- adaptive moment estimation.

decay: learning rate decay :-

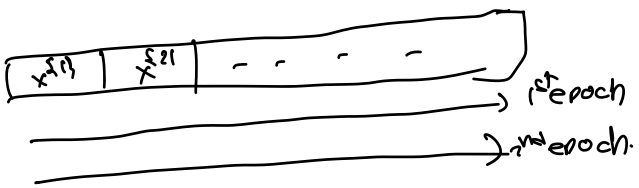
- slowly reduce learning rate overtime.

64, 128 mini batch



α - without decay

α - with decay.



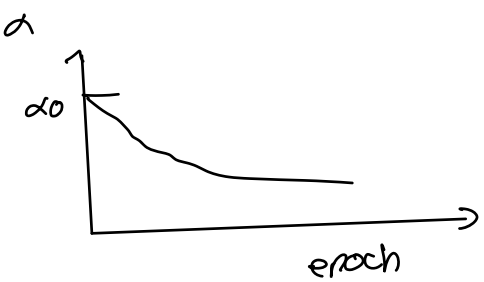
1 epoch = 1 pass through data.

$$\alpha = \frac{1}{1 + (\text{decay rate}) \times \text{epoch number}} \times \alpha_0$$

epoch $\alpha_0 = 0.2$ decay rate = 1

$\frac{0.1}{1 + (1)(1)}$

epoch	α
1	0.1
2	0.067
3	0.05
4	0.04

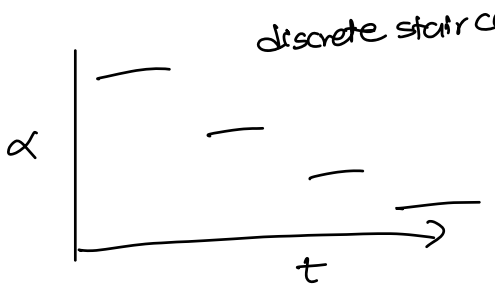


exponential decay α

$$\alpha = 0.95^{\text{epochnum}} \cdot \alpha_0$$

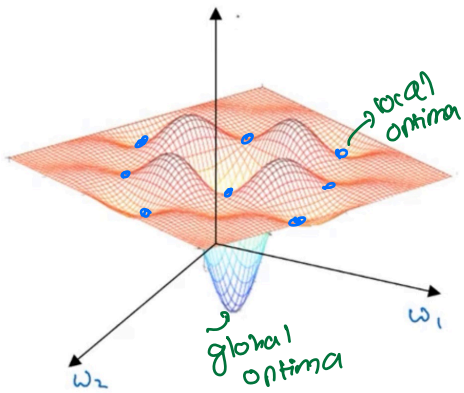
$$\alpha = \frac{k}{\sqrt{\text{epochnum}}} \cdot \alpha_0$$

$$\alpha = \frac{k}{\sqrt{t}} \cdot \alpha_0 \quad (t - \text{mini batch iter})$$



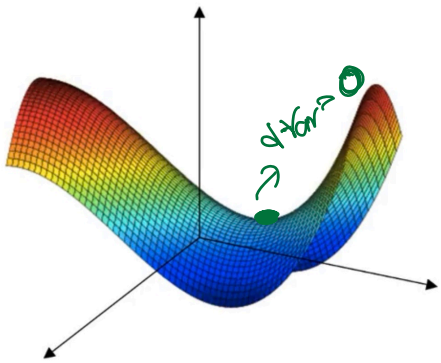
Manual decay is also possible.

dec 10: Problem of local optima:-



- most point where gradient zero are not like these.

Saddle point

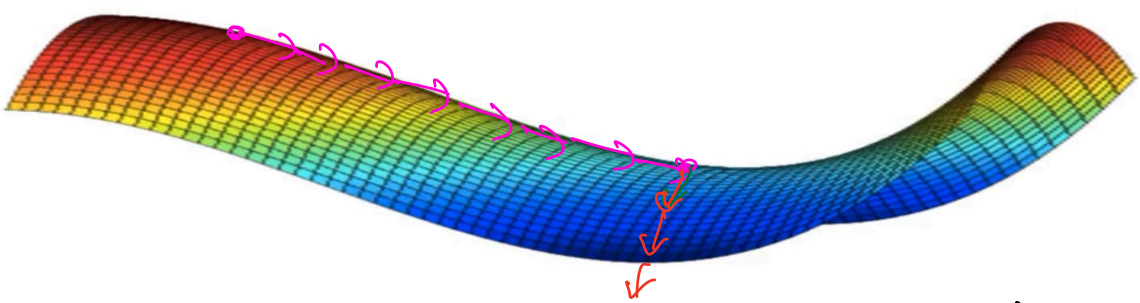


- This is more important problem.

if n = 20000 dim

↻ all should be like this 2^{-20000}

Problem of plateaus:-



- unlikely to get stuck in bad local optima
- plateaus can make learning slow
- Adam RMSprop / momentum can speed up GD