Optimization Algorithms - Coursera

SGD
Momentum
RmsProp
Adam
} Optimization methods.

→ Use Random mini batches to accelerate the convergence and improve the optimization.

Batch Gradient Descent vs. Mini Batch Gradient Descent

Suppose records are 5000000
→ 5000 * 1000 each batches

for each batch $t = 1, \ldots 5000$
{ Forward prop
→ Compute Activation fns → (Vectorized implementation on 1000 samples)

Compute cost fn $fn(x^{\{t\}}, y^{\{t\}})$

$$J^{(t)} = \frac{1}{1000} \sum_{i=1}^{l} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \times 1000} \sum_{l} \|w^{[l]}\|_2^2$$

Back propogation to compute gradients wrt $J^{(t)}$ using $x^{\{t\}}, y^{\{t\}}$

$$w^{[l]} := w^{[l]} - \alpha dw^{[l]}, \quad b^{[l]} := b^{[l]} - \alpha db^{[l]}$$

}

1 pass thru your training set (using mini batch gradient descent) — Doing 1 'epoch' of training.

'epoch' → One single pass thru the ~~gradient descent~~ training set.

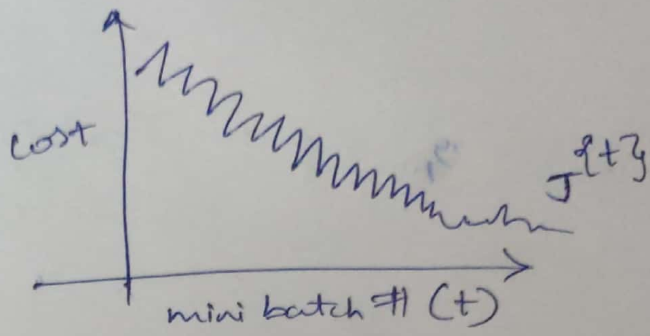
In Batch GD — 1 epoch allows us to take 1 gradient descent step

In mini Batch GD — 1 epoch allows us to take 5000 Gradient descent steps.

You might want to have another for loop to include ~~many~~ multiple passes. thru the training set. [i.e many epochs]

Batch Gradient Descent.

mini-batch gradient descent.



Training process of both.

Choosing your mini batch size (See snapshot)

Typical mini-batch sizes:

64, 128, 256 . . . . 2^n

make sure. all $X^{\{t\}}$, $Y^{\{t\}}$ fits in CPU/GPU memory.

Exponentially weighted averages.

also called

Exponentially weighted moving averages

Moving average is calculated as,

$V_0 = 0$

$V_1 = 0.9 V_0 + 0.1 \theta_1$

$V_2 = 0.9 V_1 + 0.1 \theta_2$

$\vdots$

$V_t = 0.9 V_{t-1} + 0.1 \theta_2$

$\boxed{V_t = \beta V_{t-1} + (1-\beta)\theta_t}$ → Formula to implement Exp. moving Avg

Here $\beta = 0.9$, V_t is approximately average over $\approx \frac{1}{1-\beta}$ day's temp.

So if.

$\beta = 0.9$: $\approx$ 10 days

$\beta = 0.98$: $\approx$ 50 days → more smooth line (since we are averaging on much larger window)

$\beta = 0.5$: $\approx$ 2 days → more noise

Scanned with CamScanner

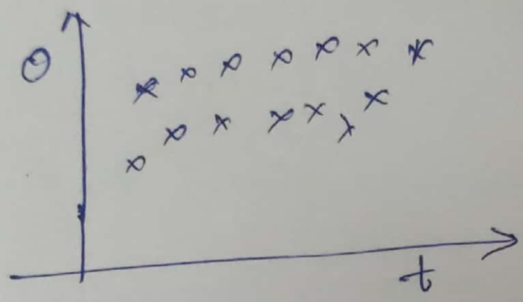
Understand Exponentially weighted Averages.

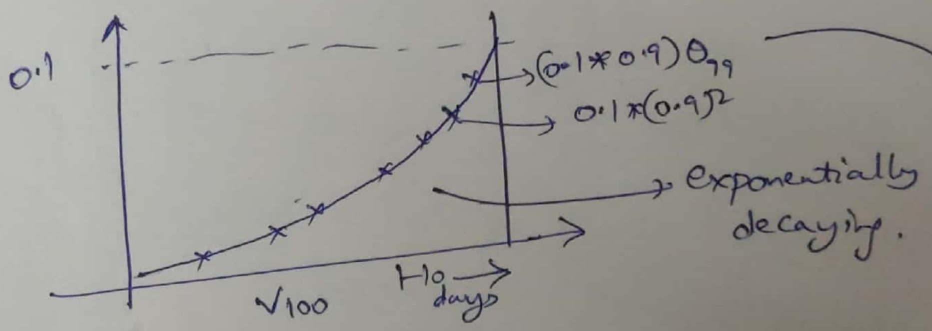
$$V_t = \beta V_{t-1} + (1-\beta)\theta_t$$

$$V_{100} = 0.9 V_{99} + 0.1\,\theta_{100}$$
$$V_{99} = 0.9 V_{98} + 0.1\,\theta_{99}$$
$$\vdots$$

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

One way to show this in pictures





$0.1,\ 0.1*0.9,\ 0.1*(0.9)^2 \ldots \rightarrow$ all these add up to 1 upto a detail called $\boxed{\text{Bias correction}}$.

How many days is this averaging over??? (refer to previous Page)

i.e its averaging over 10 days $\left(\frac{1}{1-\beta}\right)$

$$0.9^{10} \approx 0.35 \approx \frac{1}{e}$$

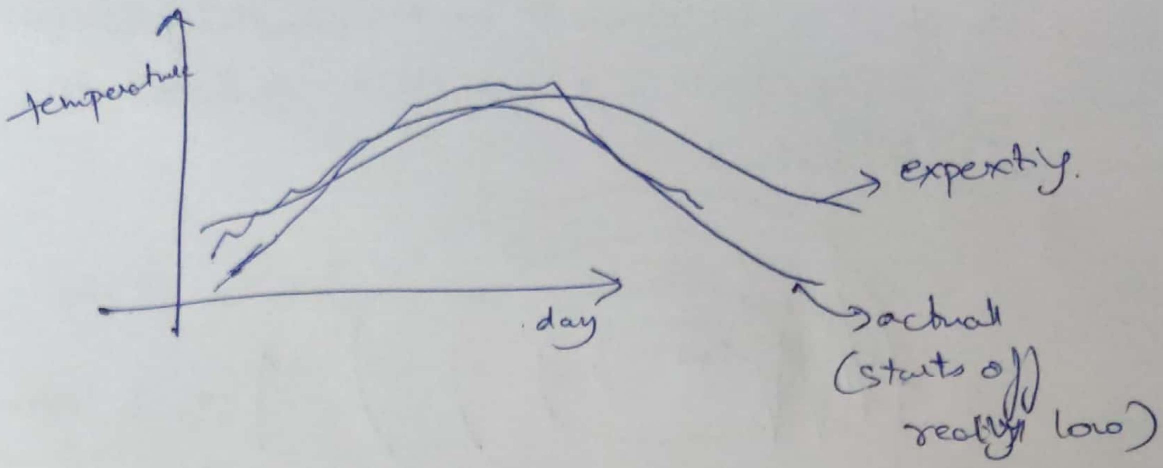
So it takes 10 days for this height to decay over $\frac{1}{3}$rd.

****

$$\boxed{(1-\epsilon)^{1/\epsilon} = \frac{1}{e}}$$

Scanned with CamScanner

Bias Correction in exponentially weighted averages.



$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

$v_0 = 0$

$v_1 = 0.98 v_0 + 0.02\theta_1$

$v_1 = 0.02\theta_1$

$v_2 = 0.98(0.02\theta_1) + 0.02\theta_2$ → very slow start to the weighted averages.

to counter this, use $\frac{v_t}{1-\beta^t}$ as v_t

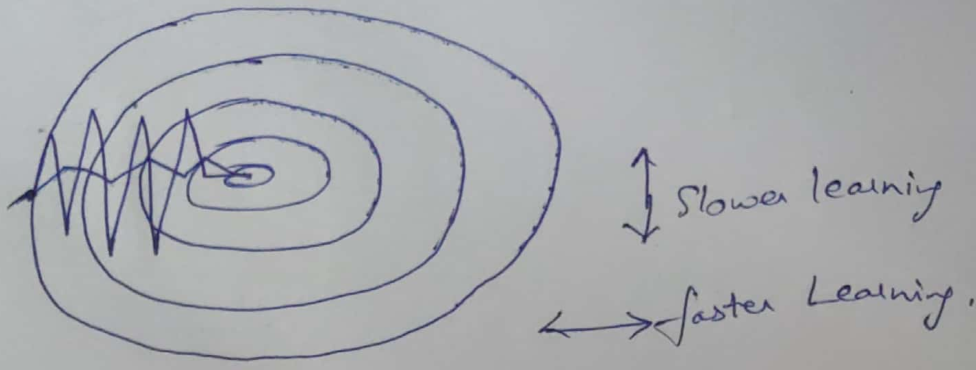
for example $t=2$:

$$\frac{v_2}{1-(0.98)^2} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$

This will give it a better start.

Gradient descent with Momentum.

The basic idea is to compute an exponentially weighted average of your gradients and use that gradient to update your weights instead.




→ ball rolling down a Bowl.

Implementation details.

On iteration t :

(prevents from speeding up) Compute dW, db on the current mini-batch.

friction

$V_{dw} = \beta V_{dw} + (1-\beta) dw$ → acceleration

velocity

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

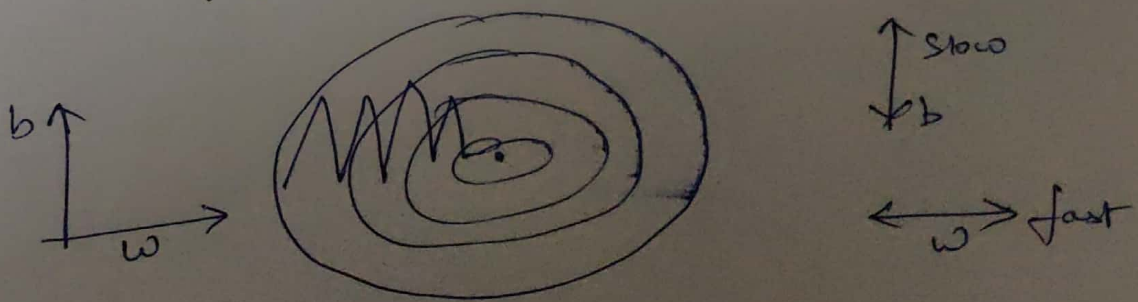
$W = W - \alpha V_{dw}, \quad b = b - \alpha V_{db}$

Hyperparameters: α, β, $\underbrace{\beta = 0.9}_{\text{Common value we use}}$

i.e Average over the last 10 gradients iterations (~~10 iterations~~)

RMSProp

Root Mean Square prop.



On iteration t:

Compute dw, db on current mini-batch

$S_{dw} = \beta S_{dw} + (1-\beta) dw^2$ element-wise squaring operation

small

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

$\underbrace{w := w - \alpha \frac{dw}{\sqrt{S_{dw}}}}_{\text{Updates } \cancel{\text{too}} \text{ in horizonatal direction is large.}}$, $\underbrace{b = b - \alpha \frac{db}{\sqrt{S_{db}}}}_{\text{So Updates in vertical direction is small}}$

This is keeping an exponentially weighted average of the square of the derivatives.

Adam Optimization Algorithm.

Momentum + RMSProp

While implementing Adam, we will need to add bias correction for V_{dw}, V_{db}, S_{dw}, S_{db}.

finally,

$$w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected} + \epsilon}}, \quad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected} + \epsilon}}$$

Hyper parameters

α: needs to be tuned

$\beta_1 : 0.9 \rightarrow dw$

$\beta_2 : 0.99 \rightarrow dw^2$

$\epsilon : 10^{-8}$

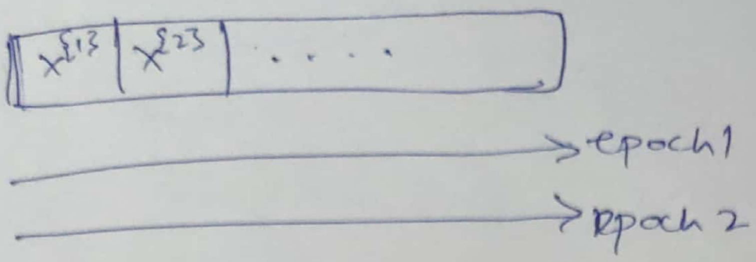
Use these default values

[Adam - Adaptive moment Estimation]

Scanned with CamScanner

Learning Rate decay

↳ To slowly reduce learning rate over time.



$$\alpha = \frac{1}{1 + \text{decay rate} * \text{epoch\_num}} \alpha_0$$

Decay rate is another hyperparameter to tune.

Epoch	α
1	0.1
2	0.067
3	0.05
4	0.04

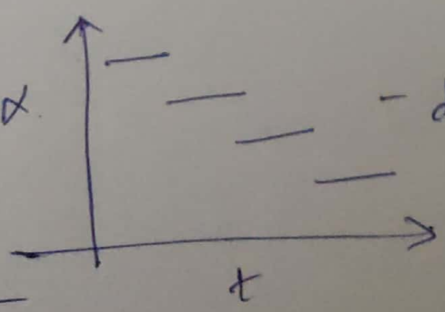
(Decays)

$\alpha_0 = 0.2$
decay.rate = 1

Formula

$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ — exponential decay

$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$



— discrete staircase

manual decay { watch your model as its training and if learning rate has slowed down, manually increase α.