Week 2 c2

Batch vs. mini-batch gradient descent

X, Y $\quad$ $X^{\{t\}}, Y^{\{t\}}$

Vectorization allows you to efficiently compute on m examples.

$$X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}\ (n_x, 1000)} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}\ (n_x, 1000)} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5,000\}}\ (n_x, 1000)}]$$

(n_x, m)

$$Y = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}\ (1, 1000)} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\ (1, 1000)} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5,000\}}\ (1, 1000)}]$$

$(1, m)$

What if $m = 5,000,000$?
5,000 mini-batches of 1,000 each
Mini-batch t: $X^{\{t\}}, Y^{\{t\}}$

$x^{(i)}$
$z^{[l]}$
$X^{\{t\}}, Y^{\{t\}}$.

Mini-batch gradient descent

- Faster than batch g.d. For one iteration , not 1 epoch
- mini batch is of 1000 egs for tr set of size 5000000
- X{t},Y{t} - tr set and label vector set for mini batch 't'
- Shape of X{t} , Y{t} is (n_x, 1000) and (1,1000) resp.
- While X and Y was (n_x , m) and (1, m) resp., cols change

repeat { for $t = 1, \dots, 5000$ {

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

$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$
$A^{[1]} = g^{[1]}(Z^{[1]})$
$\vdots$
$A^{[L]} = g^{[L]}(Z^{[L]})$

} Vectorized implementation (1000 examples) → Vectorised impltn

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|W^{[l]}\|_F^2$. (from $X^{\{t\}}, Y^{\{t\}}$.)

Backprop to compute gradients wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$) — gradient wrt

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$, $b^{[l]} := b^{[l]} - \alpha db^{[l]}$

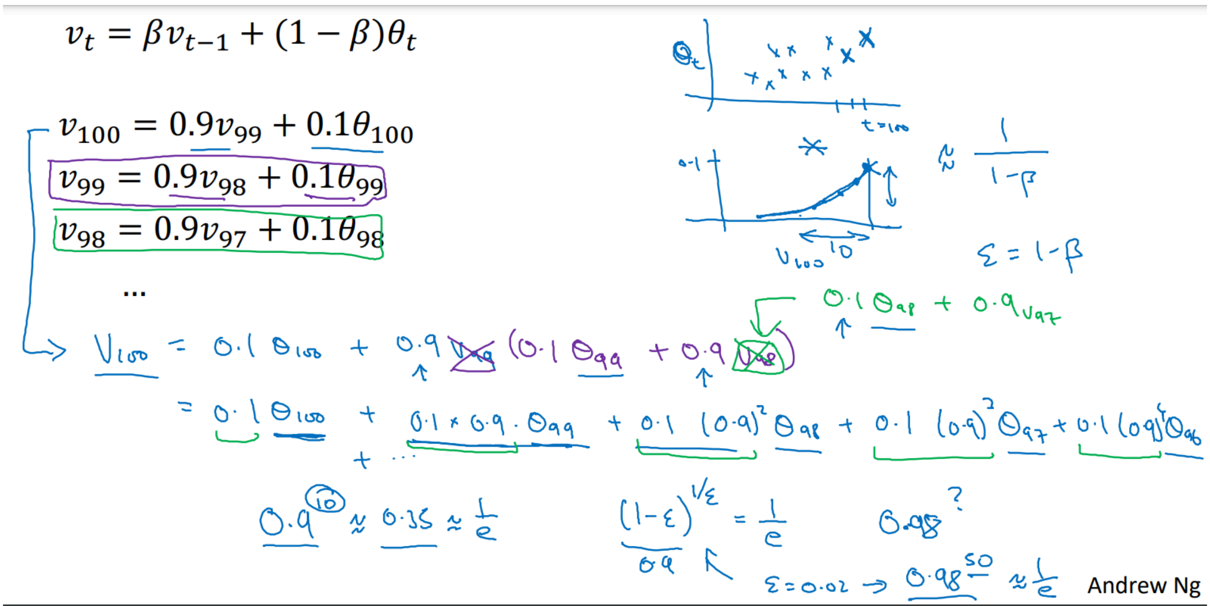
}
}

The for loop runs over a range of X{t}.shape[1] or no. of epochs

Here, each epoch takes 1000 gradient descent, which was 1 G.D. for batch G.D.
X{t},Y{t} run at same time 5000 times

Refer *dl* file for next lec notes

Implementing Exponentially Weighted avg

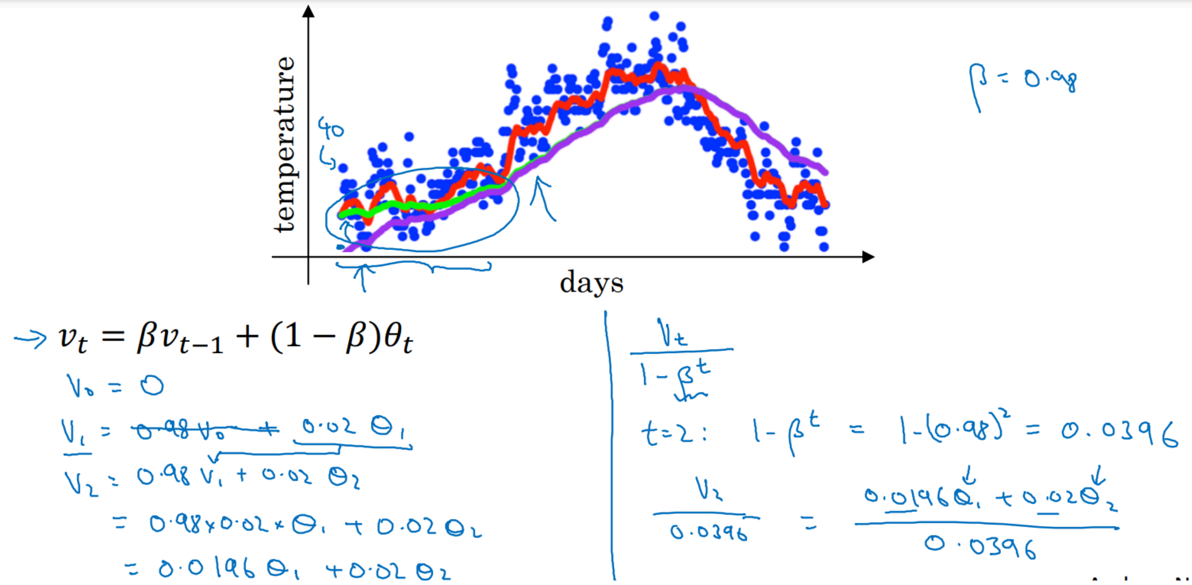


The v_100 is simplified to show the weights that temp hold as days pass,
As seen , from day 100 to day 96, the coeff decrease - results in exponentially decaying function [/curve - shown in the img]
Sum of all coeff equals 1 almost,i.e., bias correction
How many days temperature is this averaging over? I.e., 1/1-beta in terms of epsilon.
Here, 0.9^10 is 1/e, so 10 days. similarly, if beta = .98, then its 50 days, cus, 0.98^50 = 1/e . Basically, 1/e = 1/1-beta

So for implementing simple 1 line code is used = the formula of v_t
And, v_0 = 0

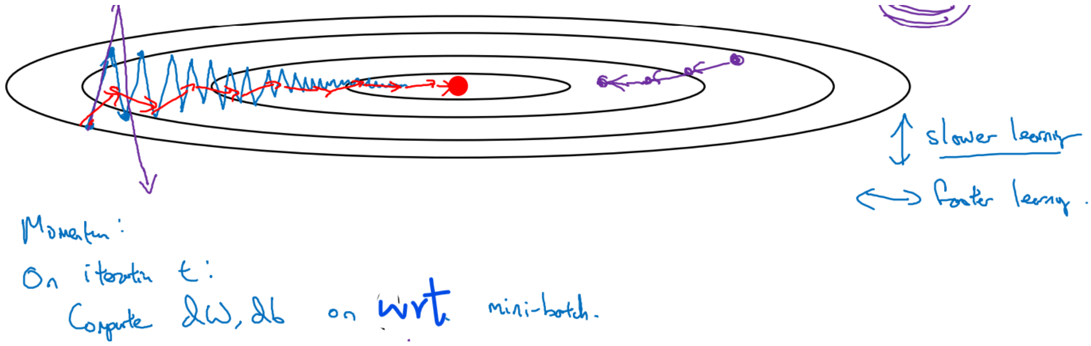
Bias correction
If beta = .98, the initial phase of the curve seems closer to x-axis

To fix this,   v_t /= 1 - beta



Purple curve → beta = .98

Gradient Descent with Momentum



The Red Dot denotes the minimum
the zigzag pattern starts from a gradient which tends to have more noise and less progress with Each titration
The oscillations cause slower gradient doesn't and a larger learning rate can't be used
Need of Slower learning vertclly, cus the learning isn’t in one dirn , and avg tends to 0
hrzlly, all pts are heading right so, derivatives inc

Beta =0.9 normally
[also , mistake ‘wrt’ must be replace by ‘current’]
Use the lhs method from the below method of taking (1 - beta) → not supposed to be neglected. Initialise vdw, vdb to 0

$v_{dw} = 0$, $v_{db} = 0$

On iteration t:

Compute dW, db on the current mini-batch

$$v_{dW} = \beta v_{dW} + \cancel{(1-\beta)}dW$$

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

$$v_{dw} = \beta v_{dw} + dW$$

$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

$$\cancel{\frac{v_{dw}}{1-\beta^t}}$$

Hyperparameters: α, β $\qquad \beta = 0.9$

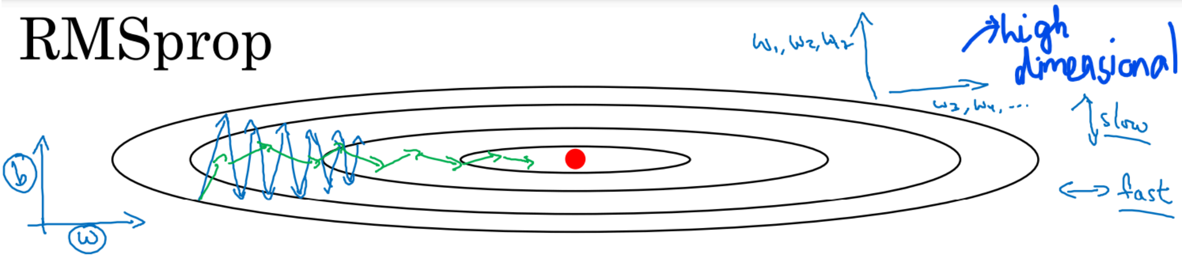
average over last ≈10 gradients

Andrew Ng

Rhs formula [without (1 - beta)] tends to scale vdw ,vdb corresponding to (1 - beta) and updation leads to alpha change by corresponding to (1 - beta) thus tuning hyperparameters
Instead use lhs
\
The updation formula contain v instead of derivative
Dimensions of dw and w are same, Ily, db and b

The calculation of J takes long , then if taking smaller J vals allowed, then
Not to increase data size
Normalise the data
better random initialisation
Use g.d. With momentum

RMSprop



On iteration t:

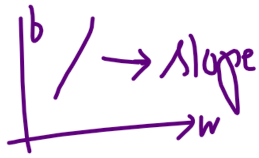
Compute dW, db on current mini-batch

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)\, dW^2$ ← small (element-wise)

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

$W := W - \alpha \dfrac{dW}{\sqrt{S_{dW}+\varepsilon}}$ $b := b - \alpha \dfrac{db}{\sqrt{S_{db}+\varepsilon}}$

$\varepsilon = 10^{-8}$



The steep slope bends more towards b , so, db is larger , and so is db^2. Note the squaring is element wise
To Balance this the updation formula divides db by root of s_db

To avoid having zero in the denominator Epsilon is added

Note - rms prop formulae have subscript 2 Associated with b.
Not there in Formulae of momentum
On Applying rms prop, we get the Green zig zag line

Adam optimization algorithm

$V_{dW} = 0,\ S_{dW} = 0.\ V_{db} = 0,\ S_{db} = 0$

On iteration t:

Compute dW, db using current mini-batch

$V_{dW} = \beta_1 V_{dW} + (1-\beta_1)\, dW$, $V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db$ ← "momentum" β_1

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)\, dW^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db$ ← "RMSprop" β_2

yhat = np.array([.9, 0.2, 0.1, .4, .9])

$V_{dW}^{corrected} = V_{dW}/(1-\beta_1^t)$, $V_{db}^{corrected} = V_{db}/(1-\beta_1^t)$

$S_{dW}^{corrected} = S_{dW}/(1-\beta_2^t)$, $S_{db}^{corrected} = S_{db}/(1-\beta_2^t)$

$W := W - \alpha \dfrac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}}+\varepsilon}$ $b := b - \alpha \dfrac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\varepsilon}$

The corrected vals formula has beta raised to t, i.e.,eg. - the day no.

So basically Adam Optimisation algorithm is combination of momentum and RMs prop

Hyperparameters choice:

→ α : needs to be tune

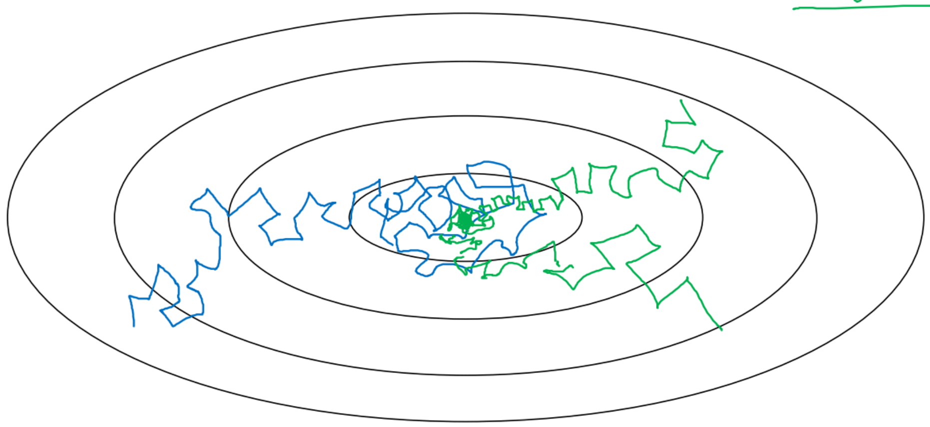
→ β_1 : 0.9 → (dw)

→ β_2 : 0.999 → (dw^2)

→ ε : 10^{-8}

Learning rate decay

Slowly reduce α



Learning rate decay

1 epoch = 1 pass through data.

$$\alpha = \frac{1}{1 + \underline{\text{decay-rate}} * \text{epoch-num}} \alpha_0$$

If alpha not = 0.2 and Decay rate = 1

Epoch	α
1	0.1
2	0.067
3	0.05
4	0.04

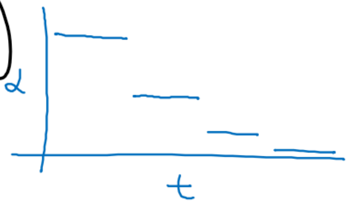
Its an Exponential decrease

Other learning rate decay methods

formula

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

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

3. 

discrete staircase

4. Manual decay.

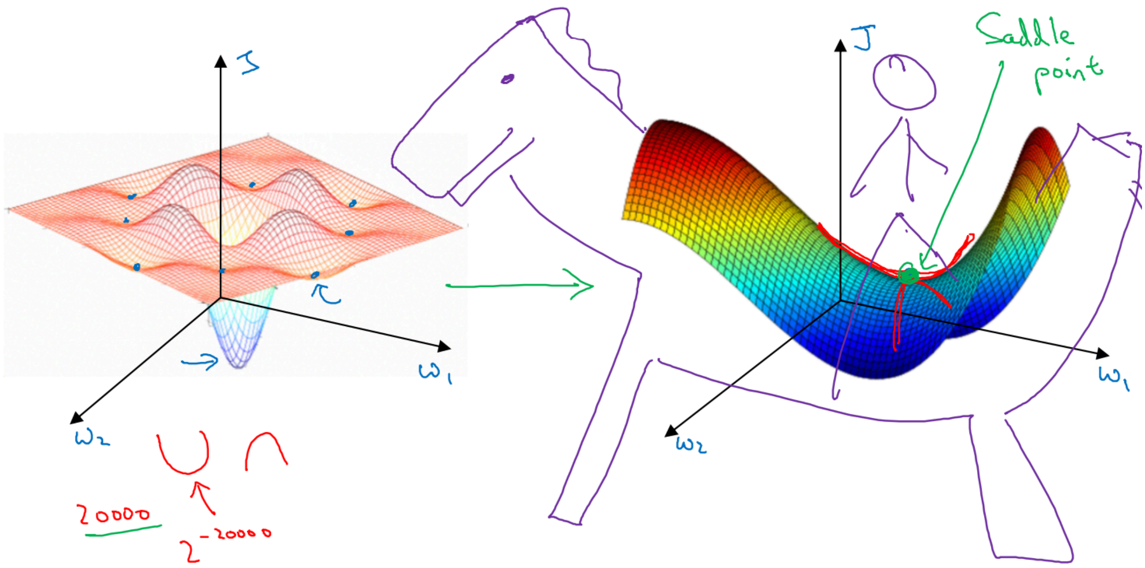
Problem faced by optimization algorithms - refer next image

flat surfaces where Global optimum isn't present are the problem earlier it was that local optimums are the problem and the first graph Where lots of local optimums are present on flat surfaces

the second graph shows actual problem - saddle points.

local optimum can never be a problem for costfunction is defined over high dimensional space ,i.e.,cost function depends on a large number of parameters or variables - typically the weights and biases

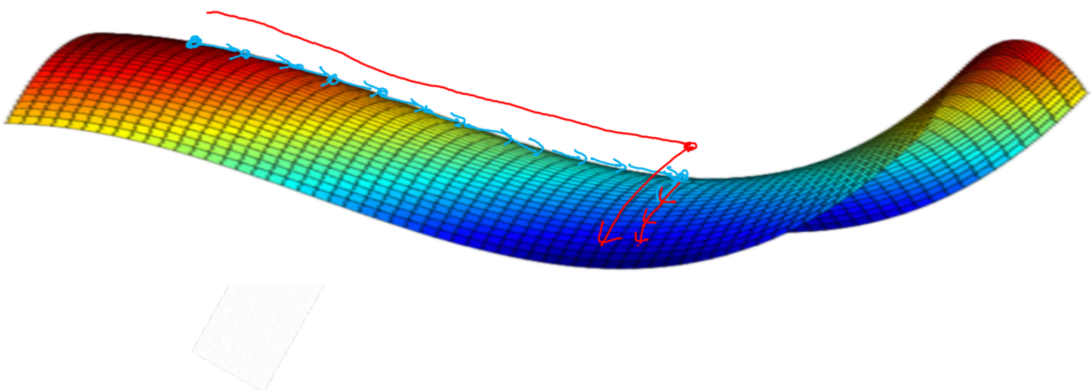
Local optima in neural networks



A saddle point is a point where the gradient is zero, but the function can have both convex and concave directions .As global optimimum is the main goal, The gradient supposed to move to the Global one but it gets stuck here, on saddle of the flat surface

Next prb

Problem of plateaus



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow