course 2 week 2

Optimization Algorithms:-

Deep learning works better in big data (huge dataset)

Batch vs minibatch gradient descent

$X = [x^{(1)}\ x^{(2)}\ x^{(3)}\ .\ .\ .\ .\ .\ x^{(m)}]$

$\rightarrow (n_x, m)$ $n_x \rightarrow$ size of single training example ($x^{(i)}$)

$m \rightarrow$ number of traing example

$Y = [y^{(1)}\ y^{(2)}\ y^{(3)}\ y^{(4)}\ .\ .\ .\ .\ .\ .\ y^{(m)}]$

Vectorization allows you to effeciently compute on m examples.

But if m is verly large then it's very time consuming

To make training more faster we make baby training set (mini-batch) from the huge training set (m).

Let's say we have 5,000,000 training example so we will create 5000 minibatch from them each having 1000 training example.

$$X = [\underbrace{x^{(1)}x^{(2)} \dots x^{(1000)}}_{X^{\{1\}} \rightarrow (n_x, 1000)} \mid \underbrace{x^{(1001)}\ x^{(1002)} \dots x^{(2000)}}_{X^{\{2\}} \rightarrow (n_x, 1000)} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5000\}}}]$$

$$Y = [\underbrace{y^{(1)}y^{(2)} \dots y^{(1000)}}_{y^{\{1\}}} \mid \underbrace{y^{(1001)}\ y^{(1002)} \dots y^{(2000)}}_{y^{\{2\}}} \mid \dots \mid \underbrace{\dots y^{(m)}}_{y^{\{5000\}}}]$$

Batch gradient descent $\rightarrow$ works on entire training set then update the parameter.

mini-batch " " $\rightarrow$ works on subset of the training set & update the parameter.

Mini batch gradient descent:-

For $t = 1 \dots 5000$

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

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

vectorize implementation (1000 example).

Compute cost:

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

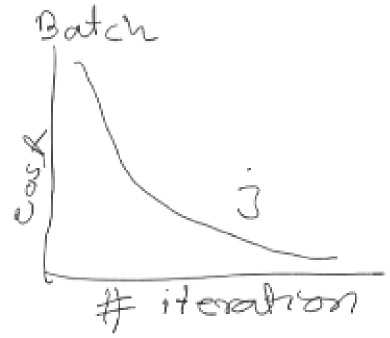
Back prop to compute gradient cost $J^{\{t\}}$ (using $X^{\{t\}}, Y^{\{t\}}$)

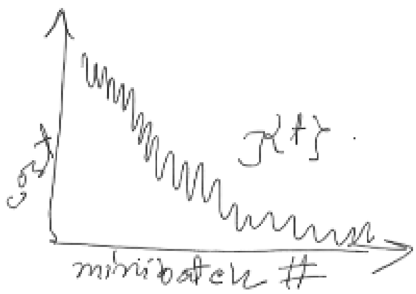
update parameter:

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

end for; → 1 epoch → single pass through the training set.

Understanding minibatch



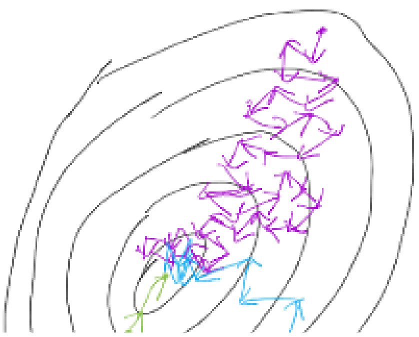


We see the noise in cost reduction because some of the minibatch (t) might contain more noise in the training.

Size of minibatch:-

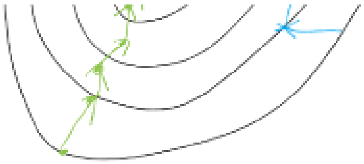
if mini-batch size = m : batch gradient descent $\{X^{\{1\}}, Y^{\{1\}}\} = \{X, Y\}$

if mini-batch size = 1 : Stochastic " " → Every example is a minibatch



Low noise, large step, keep merging to min^m.

most of the time towards minimum but some times go in wrong direction. Extremerely noisy. Won't ever converge. It will wander



around the minimum

stocastic	mini batch (Practie)	Batch.
we loose the speed up as we can't use the vectorization.	→ faster we can also implement vectorization → can make progress without processing entire training set.	takes too much time per iteration.

Choosing minibatch size :-

if small training set ($m \le 2000$) use batch gradient descent

typical mini batch size :-

64 , 128 , 256 , 512

2^6 , 2^7 , 2^8 , 2^9

| if size is power of 2 computer works faster.

make sure mini batch fits CPU/GPU memory

It's one of the hyperparameter which needs to be searched.

Exponentially weighted average :-

temperature in London :-

$\theta_1 = 40°F$
$\theta_2 = 49°F$
$\vdots$
$\theta_{180} = 60°F$
$\theta_{181} = 56°F$
$\vdots$

$v_0 = 0$

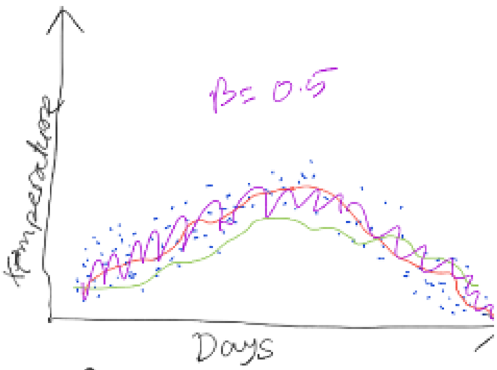
$v_1 = 0.9 v_0 + 0.1 \theta_1$

$v_2 = 0.9 v_1 + 0.1 \theta_2$

$v_3 = 0.9 v_2 + 0.1 \theta_3$

$\vdots$

$v_t = 0.9 v_{t-1} + 0.1 \theta_t$ → Here $\beta = 0.9$



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

V_t = average of last $\frac{1}{1-\beta}$ days temperature.

$\beta = 0.9$, $\frac{1}{1-\beta} = 10$, V_t = average of last 10 days temp

$\beta = 0.98$, $\frac{1}{1-\beta} = 50$ V_t = " " " 50 " "

High value of β gives more smoother graph. Cause we are averaging more days of temperature

β is a hyperparameter.

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

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

$$V_{98} = 0.9 V_{97} + 0.1 V_{98}$$

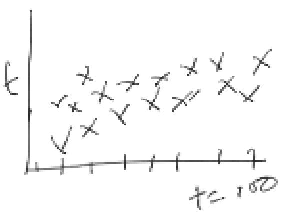
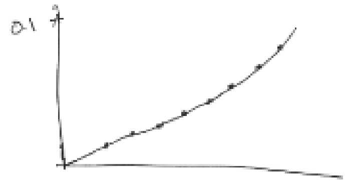
$\vdots$

$$(0.1\theta_{98} + 0.9 \cancel{V_{97}})$$

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

$$= 0.1\,\theta_{100} + 0.1 \times 0.9 \times \theta_{99} + 0.1 \times 0.9 \times 0.9 \times \theta_{98} \cdots$$

$$= 0.1\,(0.9)^0 \theta_{100} + 0.1 \times (0.9)^1 \theta_{99} + 0.1\,(0.9)^2 \theta_{98} \cdots$$

$V_{100} = \theta_t$  $*$  ← exponentially decaying functi

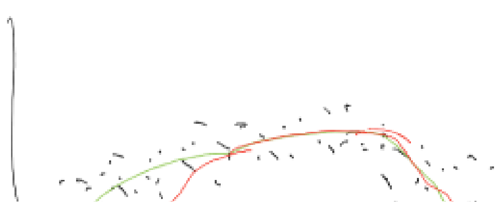
$$0.9^x = \frac{1}{e} = 0.35$$

| it takes x days to for the height of EDF decay 1/3

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

After x days the weight decays less than 1/3 which is negligible. so we say its average of x days temperature

Bias correction.



line with bias correction.

line without bias correction.

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

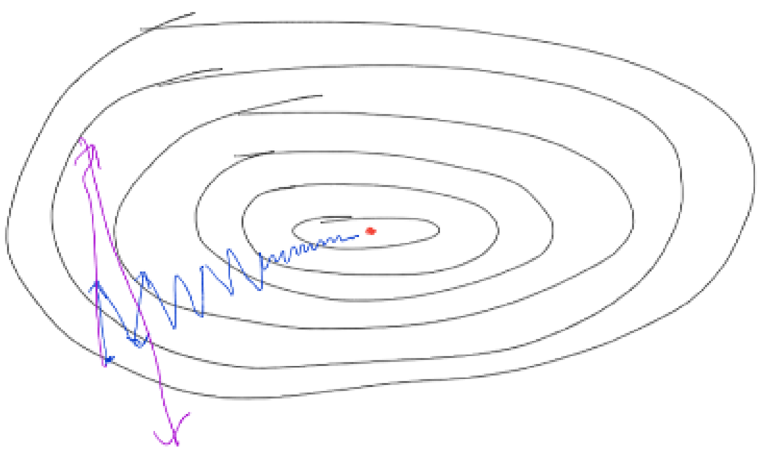
$V_0 = 0$.

$V_1 = \boxed{0.98 V_0} + 0.02\,\theta_1$

0 → Because of this term vanishes our line without bias line start from below.

to solve this issue use $\frac{V_t}{1-\beta^t}$ it will bump up the initial line. As t becomes large the denominator close to 1 so $\frac{V_t}{1-\beta^t}$ becomes V_t.

Gradient descent with momentum:-



this up and down in gradient descent takes lots of steps & prevent using larger learning rate.

Larger learning rate will overshoot the gradient descent.

so in vertical axis we want slower learning & in horizontal axis we want faster learning

momentum:-

On iteration t:-

compute dw, db on current minibatch.

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

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

$w := w - \alpha V_{dw}$.

$b := b - \alpha V_{db}$

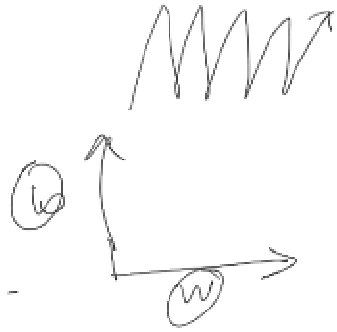
we are averaging the gradient descent curve if the curve oscillate (\/\/\/\/) then positive & negative will give us lower average implementation:-

β is a hyperparameter. but usually people use 0.9.

Bias correction isn't use in practice because after few iteration the curve fixed itself

⇒ in few implementation $(1-\beta)$ is not used instead the value of α is adjusted.

RMSprop:-



⇒ speed up gradient descent.

On iteration t:

compute dw, db on current mini-batch:-

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

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

$$w := w - \alpha \frac{dw}{\boxed{\sqrt{S_{dw}}}}$$ ← small number → increases have more effect.

$$b := b - \alpha \frac{db}{\boxed{\sqrt{S_{db}}}}$$ ← large number → Decreases – less "

Let, w is in horizontal direction & b is in vertical direction. so if derivative oscilate db will be more & dw will be less. (↑).

in practice w, b doesn't have any direction. $\frac{dw}{\sqrt{S_{dw}}}$ & $\frac{db}{\sqrt{S_{db}}}$ we are just damping the term which has more derivative(

Adam optimization algorithm:-

combination of momentum & RMSprop.

On iteration t:-

compute dw, db using current t.

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

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

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

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

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

→ it's a very small number & is added so that denominator never becomes 0.

Hyperparameter choice:-

α :- needs to be tune.

β_1 :- 0.9 (dw)

β_2 :- 0.999 (dw^2)

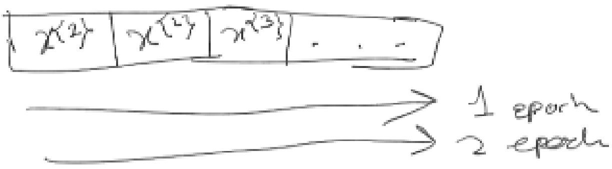
ε :- 10^{-8}

Adam ⇒ Adaptive moment estimation

Learning rate decay:-

slowly reduce learning rate over time. At initial step we can take larger steps as it converges it's step should be short or we might miss the convergence point.

1 epoch = 1 pass through data.



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

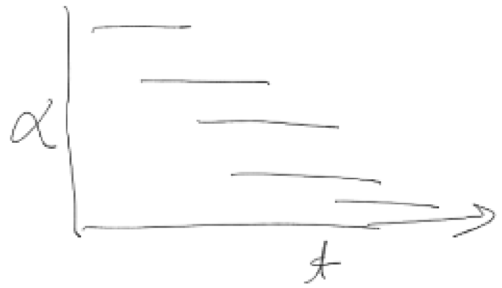
$\alpha_0 = 0.2$
decay-rate = 1

Epoch	α
1	0.1
2	0.67
3	0.05

Other decay method:-

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

$\alpha = \frac{K}{\sqrt{\text{epoch-num}}} \cdot \alpha_0 \Big/ \frac{K}{\sqrt{t}} \cdot \alpha_0$



← After some iteration we reduce α by ½
this is called discrete staircase

Local optimum :-

⇒ plateaus is a region where derivative is almost zero

⇒ in higher dimension instead of local optimum we get saddle point.